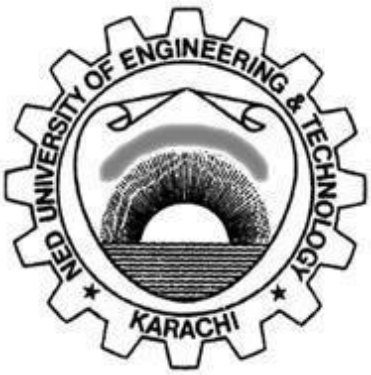
Workbook

Software Quality Engineering (SE-309)



Name Musadique Hussain

Roll No SE-21031

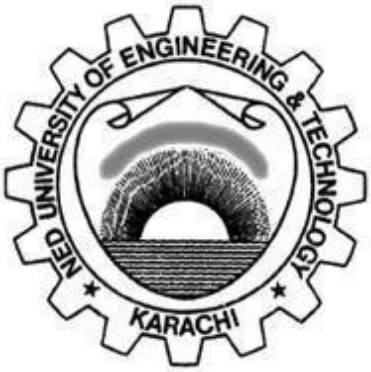
Batch 2021

Year 2024

Department Software Engineering

Workbook

Software Quality Engineering (SE-309)



Prepared by
Engr. Sana Fatima
Lecturer- SE

Approved by
Chairman

Department of Software Engineering

Table of Contents

14	Desktop App Testing: Test Notepad Application through Katalon studio	68	

Lab 01: Introduction to Software Testing

Software: Software is a set of instructions to perform some task. Software is used in many applications of the real world. Some of the examples are

- Application software, such as word processors
- Firmware in a embedded system
- Middleware, which controls and co-ordinates distributed systems
- System software such as operating systems
- Video Games
- Websites

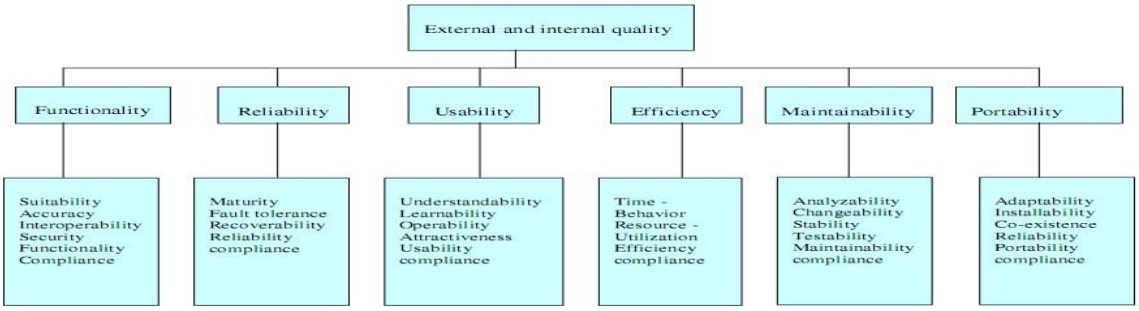
All of these applications need to run without any error and provide a quality service to the user of the application. In this regard the software has to be tested for its accurate and correct working.

In order to make sure the released software is safe and functions as expected, the concept of **software quality** was introduced. It is often defined as "*the degree of conformance to explicit or implicit requirements and expectations*". These so-called explicit and implicit expectations correspond to the two basic levels of software quality:

- **Functional** – the product's compliance with functional (explicit) requirements and design specifications. This aspect focuses on the practical use of software, from the point of view of the user: its features, performance, ease of use, absence of defects.
- **Non-Functional** – system's inner characteristics and architecture, i.e. structural (implicit) requirements. This includes the code maintainability, understandability, efficiency, and security.

Software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance).Software quality is a multidimensional quantity and is measurable. To do this, we need to divide and measure software quality in terms of quality attributes:

Software Quality Attributes:



ISO-9 126 (ISO, 2001) provides a hierarchical framework for quality definition, organized into quality characteristics and sub-characteristics

Functionality: *A set of attributes that bear on the existence of a set of functions and* their specified properties. The functions are those that satisfy stated or implied needs. The sub-characteristics include: - Suitability - Accuracy Interoperability - Security

Reliability: *A set of attributes that bear on the capability of software to maintain* its level of performance under stated conditions for a stated period of time. The sub-characteristics include: - Maturity - Fault tolerance - Recoverability

Usability: *A set of attributes that bear on the effort needed for use, and on the* individual assessment of such use, by a stated or implied set of users. The sub-characteristics include: - Understandability - Learnability - Operability

Efficiency: *A set of attributes that bear on the relationship between the level of per*formance of the software and the amount of resources used, under stated conditions. The sub-characteristics include: - Time behavior - Resource behavior

Maintainability: A set of attributes that bear on the effort needed to make specified modifications. The sub-characteristics include: - Analyzability - Changeability - Stability - Testability

Portability: *A set of attributes that bear on the ability of software to be transferred* from one environment to another. The sub-characteristics include: - Adaptability - Installability - Conformance – Replaceability

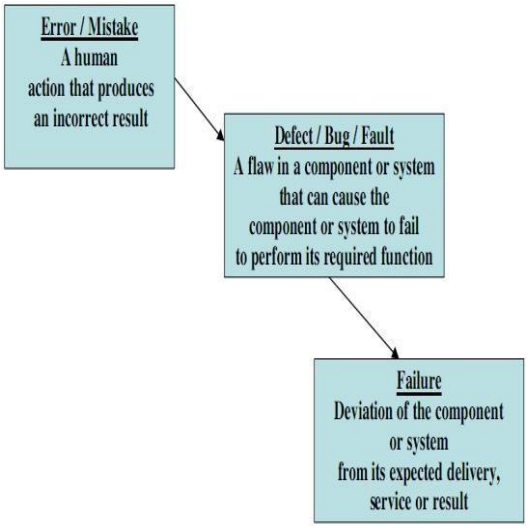
QA, QC and Testing in software development process

The structural quality of the software is usually hard to manage: It relies mostly on the expertise of the engineering team and can be assured through code review, analysis and refactoring. At the same time, functional aspect can be assured through a set of dedicated **quality management activities**, which includes quality assurance, quality control, and testing. Often used interchangeably, the three terms refer to slightly different aspects of software quality management. Despite a common goal of delivering a product of the best possible quality, both structurally and functionally, they use different approaches to this task.

	QA	QC	Testing
Purpose	Setting up adequate processes, introducing the standards of quality to prevent the errors and flaws in the product	Making sure that the product corresponds to the requirements and specs before it is released	Detecting and solving software errors and flaws
Focus	Processes	Product as a whole	Source code and design
What	Prevention	Verification	Detection
Who	The team including the stakeholders	The team	Test Engineers, Developers
When	Throughout the process	Before the release	At the testing stage or along with the development process

Functional Vs non-functional testing

Functional testing refers to tests that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".



Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security. Non-functional testing tends to answer such questions as "how many people can log in at once", or "how easy is it to hack this software".

Error, Fault and Failure:

Humans make errors in their thoughts, actions, and in the products that might result from their actions. Errors occur in the process of writing a program. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

Not all software defects are caused by coding errors. One common source of expensive defect is caused by requirement gaps, e.g., unrecognized requirements that result in errors of omission by the program designer. A common source of requirement's gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Errors – Examples

- Incorrect usage of software by users
- Bad architecture and design by architects and designers
- Bad programming by developers
- Inadequate testing by testers
- Wrong build using incorrect configuration items by Build Team Member

Fault - Examples

- A fault is the manifestation of one or more errors
- An incorrect statement

- Wrong data type
- Wrong mathematical formula in design document
- Missing functionality in the system

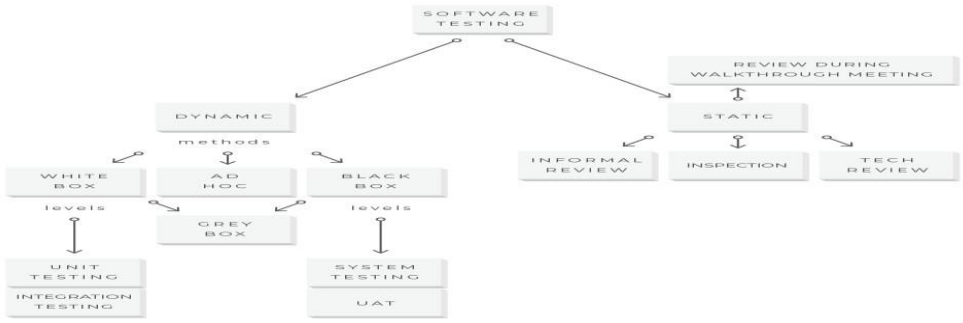
Failure

A failure occurs when a faulty piece of code is executed leading to incorrect state that propagates to the program's output. The following figure tells us how Error made by human will result in failure of the software.

Software Testing Objectives: Testing is done to fulfill certain objectives:

- To discuss the distinctions between validation testing and defect testing
- To describe the principles of system and component testing
- To describe strategies for generating system test cases
- To understand the essential characteristics of tool used for test automation
- To find or prevent defects
- To determine that software products satisfy specified requirements
- Ensuring that a system is ready for use
- Gaining confidence that it works
- Providing information about the level of quality
- Determining user acceptability

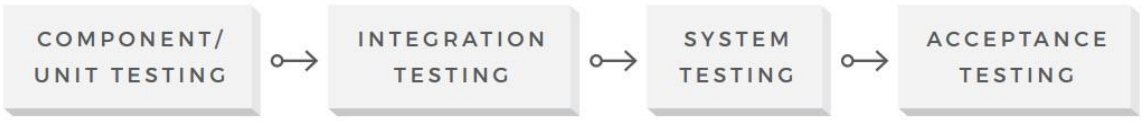
Software Testing Types:



- **Static testing** initially examines the source code and software project documents to catch and prevent defects early in the software testing life cycle. Also called non-execution technique or verification testing, static testing could be performed as inspections, informal and technical reviews, or reviews during walkthrough meetings. Informal review is a cheap testing variant that a QA analyst can conduct anytime during the project. Inspection, also called a formal review, is planned and controlled by the moderator. During the review meeting, errors found by QA analysts are discussed and documented in the review report.
- **Dynamic testing** :software is tested during execution. This whitepaper has the most focus on the dynamic testing process as a practical and most commonly used way to validate code behavior.

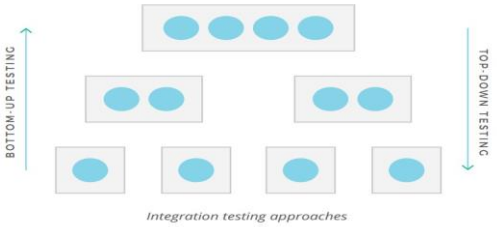
Dynamic testing can be described by methods, levels, and types of underlying QA activities. Let's have a closer look at this segment of the dynamic testing process.

Software testing levels describe stages of software development when testing is conducted. That said, there are four progressive testing levels based on the area they focus on the software development process: unit testing, integration testing, system testing, and user acceptance testing (UAT).



- **Component/Unit Testing**: The smallest testable part of the software system is often referred to as a unit. Therefore, this testing level is aimed at examining every single unit of a software system in order to make sure that it meets the original requirements and functions as expected. Unit testing is commonly performed early in the development process by the engineers themselves, not the testing team.
- **Integration Testing**: The objective of the next testing level is to verify whether the combined units work well together as a group. Integration testing is aimed at detecting the flaws in the interactions between the units within a module. There are two main approaches to this testing: bottom-up and top-down methods. The bottom-up integration testing starts with unit tests, successively increasing the complexity of the software modules under test. The top-down method takes the opposite approach, focusing on high-level combinations first and examining the simple ones later.
- **System Testing**: At this level, a complete software system is tested as a whole. This stage serves to verify the product's compliance with the functional and technical requirements and overall quality standards. System testing should be performed by a highly professional testing team in an environment as close to the real business use scenario as possible.
- **User Acceptance Testing**: This is the last stage of the testing process, where the product is validated against the end user requirements and for accuracy. This final step helps the team decide if the product is ready to be shipped or not. While small issues should be detected and resolved earlier in the process, this testing level focuses on overall system quality, from content and UI to performance issues. The acceptance stage might be followed by an alpha and beta testing, allowing a small number of actual users to try out the software before it is officially released.

Software testing methods are the ways the tests are conducted. They include black box testing, white box testing, and gray box testing, and ad hoc testing.



Integration testing approaches

- **Black Box Testing:** This method gets its name because a QA engineer focuses on the inputs and the expected outputs without knowing how the application works internally and how these inputs are processed. The

purpose of this method is to check the functionality of the software making sure that it works correctly and meets user demands. This method can be applied to any testing level but is used mostly for system and user acceptance testing. A QA specialist doesn't consider the internal processes of the product while conducting a test

- **White Box Testing:** Unlike black box testing, this method requires profound knowledge of the code as it entails testing of some structural part of the application. Therefore, generally, the developers directly involved in writing code are responsible for this type of testing. The purpose of white box testing is to enhance security, the flow of inputs/outputs through the application, and to improve design and usability. This method is mainly used at the unit and integration testing levels.
- **Gray Box Testing:** This method is a combination of the previous two, since it involves testing of both functional and structural parts of the application. Using this method, an experienced tester has partial knowledge of the internal application structure and based on this knowledge can design test cases while still testing from the black-box perspective. This method is mostly applicable to the integration testing level.

Software Testing Life Cycle: Software testing life cycle identifies what test activities to carry out and when (what is the best time) to accomplish those test activities. Even though testing differs between organizations, there is a testing life cycle. Software Testing Life Cycle consists of six (generic) phases:

- Test Planning,
- Test Analysis,
- Test Design,
- Construction and verification,
- Testing Cycles,
- Final Testing and Implementation and ● Post Implementation.

Test Planning : This is the phase where the Project Manager has to decide what things need to be tested, do I have the appropriate budget etc. Naturally proper planning at this stage would greatly reduce the risk of low quality software. This planning will be an ongoing process with no end point.

Activities at this stage would include preparation of a high level test plan-(according to IEEE test plan template The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.). Almost all of the activities done during this stage are included in this software test plan and revolve around a test plan.

Test Analysis : Once test plan is made and decided upon, next step is to delve little more into the project and decide what types of testing should be carried out at different stages of SDLC, do we need or plan to automate, if yes then when the appropriate time to automate is, what type of specific

documentation we need for testing. Proper and regular meetings should be held between testing teams, project managers, and development teams, Business Analysts to check the progress of things which will give a fair idea of the movement of the project and ensure the completeness of the test plan created in the planning phase, which will further help in enhancing the right testing strategy created earlier. We will start creating test case formats and test cases itself. In this stage we need to develop Functional validation matrix based on Business Requirements to ensure that all system requirements are covered by one or more test cases, identify which test cases to automate, begin review of documentation, i.e. Functional Design, Business Requirements, Product Specifications, Product Externals etc. We also have to define areas for Stress and Performance testing.

Test Design : Test plans and cases which were developed in the analysis phase are revised. Functional validation matrix is also revised and finalized. In this stage risk assessment criteria is developed. If you have thought of automation then you have to select which test cases to automate and begin writing scripts for them. Test data is prepared. Standards for unit testing and pass / fail criteria are defined here. Schedule for testing is revised (if necessary) & finalized and test environment is prepared.

Construction and verification : In this phase we have to complete all the test plans, test cases, complete the scripting of the automated test cases, Stress and Performance testing plans needs to be completed. We have to support the development team in their unit testing phase. And obviously bug reporting would be done as when the bugs are found. Integration tests are performed and errors (if any) are reported.

Testing Cycles : In this phase we have to complete testing cycles until test cases are executed without errors or a predefined condition is reached. Run test cases --> Report Bugs --> revise test cases (if needed) --> add new test cases (if needed) --> bug fixing --> retesting (test cycle 2, test cycle 3….).

Final Testing and Implementation :In this we have to execute remaining stress and performance test cases, documentation for testing is completed / updated, provide and complete different matrices for testing. Acceptance, load and recovery testing will also be conducted and the application needs to be verified under production conditions.

Post Implementation : In this phase, the testing process is evaluated and lessons learnt from that testing process are documented. Line of attack to prevent similar problems in future projects is identified. Create plans to improve the processes. The recording of new errors and enhancements is an ongoing process. Cleaning up of the test environment is done and test machines are restored to base lines in this stage.

EXERCISE:

1. **Name the stages involved in the software testing life cycle, differentiate test planning and test analysis.**

The following are the stages which are involved in software testing life cycle are:

- Requirement Analysis
- Test Planning
- Test Case Development
- Test Design
- Test Environment Setup
- Test Execution
- Test Closure
- Test Reporting

Requirement Analysis: This stage involves understanding the software requirements to determine what needs to be tested.

Test Planning: In this stage, the test strategy and test plan are created. It involves identifying test objectives, scope, resources, timelines, and risks.

Test Case Development: Test cases are designed based on requirements and test scenarios. These test cases serve as a blueprint for executing tests.

Test design: Test design is a process that defines how testing has to be done. It involves the process of identifying the testing techniques, test scenarios, test cases, test data, and expected test results.

Test Environment Setup: The test environment is prepared with the necessary hardware, software, and network configurations required for testing.

Test Execution: Test cases are executed in the prepared test environment, and defects are identified and reported.

Test Closure: This involves evaluating if the testing objectives were met, documenting lessons learned, and preparing for future releases or projects.

Test Reporting: Test results, including metrics such as test coverage, pass/fail status, and defect analysis, are documented and communicated to stakeholders.

Differentiate of Test planning and Test Strategy

Test Planning:

- Test planning is the process of determining what to test, how to test it, and in what order.
- It involves creating a test strategy and test plan, which outline the approach to testing, resource allocation, timelines, and risk management.

- Test planning focuses on setting up the foundation for testing activities, including defining test objectives, scope, and identifying the test environment requirements.

Test Analysis:

- Test analysis involves analyzing the requirements and identifying test conditions and scenarios.
- It aims to understand the system requirements thoroughly to create effective test cases.
- Test analysis focuses on understanding what needs to be tested and how to ensure comprehensive coverage of the system's functionality and behavior.

2. **Differentiate between white box and black box testing techniques. Which technique do you prefer where you are asked to perform testing at component level and why? Explain with suitable examples.**

Black Box Testing: Black Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. Only the external design and structure are tested.

White Box Testing: White Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Implementation and impact of the code are tested.

Differentiate between Black box testing and White box Testing

Aspect	**White box Testing**	**Black Box Testing**
Focus	Internal logic and structure of the software.	External behavior and functionality of the software.

Design Documentation	Testers use design documentation to understand the system's architecture and logic.	Design documentation may or may not be available; testers rely on specifications and requirements.
Testing Approach	Tests are designed based on the understanding of how the software is implemented.	Tests are designed based on specifications, requirements, and expected behavior.
Techniques Used	Statement coverage, branch coverage, path coverage, condition coverage.	Equivalence partitioning, boundary value analysis, decision table testing, state transition testing.
Goal	Verify correctness of the code, find logic errors, ensure all code paths are tested.	Validate the software against functional requirements, identify defects in behavior, ensure it meets user expectations.
Access to code	Testers have access to the source code.	Testers do not have access to the source code.

When asked to perform testing at the component level, the preferred technique may depend on various factors such as project requirements, time constraints, and resource availability. However, if I were to choose, I would prefer white box testing at the component level for the following reasons:

Granular Testing: At the component level, having access to the internal code allows for more granular testing. This means that specific paths, conditions, and functionalities can be thoroughly tested, ensuring better coverage.

Early Detection of Defects: White box testing facilitates the early detection of defects within the component. By examining the internal logic and design, potential issues can be identified and addressed before they escalate into larger problems during integration or system testing.

Optimized Test Cases: With knowledge of the internal structure, test cases can be designed to target specific code paths and functionalities, optimizing the testing process and maximizing the effectiveness of testing efforts.

Comprehensive Coverage: White box testing techniques such as code coverage analysis enable comprehensive coverage of the component's code base, ensuring that all code paths are exercised and tested thoroughly.

Example:

Imagine a scenario where you are asked to test a sorting algorithm component. Using white box testing techniques, you would have access to the internal code of the sorting algorithm. You could design test cases to ensure that the algorithm correctly sorts various types of input data (e.g., integers, strings) and handles edge cases such as empty arrays or arrays with a single element. By examining the internal logic of the algorithm, you can verify that it behaves correctly under different conditions and efficiently handles all possible scenarios. This level of insight and control over the testing process can significantly enhance the quality and reliability of the sorting algorithm component.

3. Explain software testing levels.

Software Testing is an activity performed to identify errors so that errors can be removed to obtain a product with greater quality. To assure and maintain the quality of software and to represent the ultimate review of specification, design, and coding, Software testing is required. There are different levels of testing.

There are 4 Software Testing levels:

Unit Testing: In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that they are fit for use by the developers. It is the smallest testable part of the software.

Integration Testing: In this testing, two or more modules which are unit tested are integrated to test i.e., technique interacting components, and are then verified if these integrated modules work as per the expectation or not, and interface errors are also detected.

System Testing: In system testing, complete and integrated Software's are tested i.e., all the system elements forming the system are tested as a whole to meet the requirements of the system.

Acceptance Testing: This is a kind of testing conducted to ensure that the requirements of the users are fulfilled before its delivery and that the software works correctly in the user's working environment.

4. Write down some advantages of software testing.

The advantages of Software Testing are:

Bugs Identification: Testing helps in identifying bugs, errors, and defects in the software early in the development lifecycle, allowing for timely fixes and preventing potential issues from escalating into more significant problems later on.

Improved Quality: By systematically verifying and validating the software against its requirements, specifications, and user expectations, testing helps ensure that the software meets the desired quality standards, leading to higher customer satisfaction.

Enhanced Reliability: Testing helps in improving the reliability and stability of the software by uncovering and addressing issues related to functionality, performance, security, and other critical aspects, thereby reducing the likelihood of failures or unexpected behavior in production environments.

Cost Savings: Detecting and fixing defects during the early stages of development through testing is more cost-effective than addressing them later in the software lifecycle or after deployment. Testing helps in reducing the overall cost of development and maintenance by avoiding expensive rework and customer support.

Risk Mitigation: Testing helps in identifying and mitigating risks associated with software development, deployment, and usage. By assessing the impact and likelihood of potential risks, testing enables stakeholders to make informed decisions and implement appropriate measures to minimize risks throughout the software lifecycle.

5. **Write a test set for copying a file XYZ from folder A to folder B. (at least 5 test cases)**

Test case ID	**Description**	**Steps**	**Test Data**	**Expected Output**	**Actual Output**	**Status**
TC-001	Validate that a new folder is created	- Create a new folder 'B'	N/A	Folder should be created	Folder is created	Pass
TC-002	Check that file 'XYZ' is present in folder 'A'	-Go to the location of folder 'A' - Find file 'XYZ' in folder 'A'	N/A	File 'XYZ' should be present in folder 'A'	File 'XYZ' is present in folder 'A'	Pass
TC-003	Copy file 'XYZ' from folder 'A'	- Select file 'XYZ' -Right click on that file and click on the "copy" action	N/A	"Copy" action should be present and file should be copied.	"Copy" action is present and the file is copied.	Pass
TC-004	Paste file 'XYZ' in folder 'B'	- Go to the location of folder 'B' -Right click, and click on "paste"action	N/A	"Paste" action should be there in folder 'B' and the file should be pasted.	File is successfully pasted in folder 'B'	Pass
TC-005	Check that the file name, size and file content should be same in folder 'B'	- Go to folder 'B' - Click on the file specification and	N/A	All the properties of the file should be same	All the properties of the file are same	Pass

TC- 006	Check that the 'XYZ' file is still present in folder 'A' after it is pasted in folder 'B'	- Go to folder 'A' - Check that the 'XYZ' file is still present	N/A	File 'XYZ' should be present in folder 'A'.	File 'XYZ' is present in folder 'A'	Pass

Lab 02: Manual Testing: Test Cases Creation

/*Write Test Cases for Facebook login and signup pages

Test case/data

- A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- A test case is a pair consisting of test data to be input to the program and the expected output. The test data is a set of values, one for each input variable.
- A test set is a collection of zero or more test cases.
- A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is functioning correctly. Test cases are often referred to as test scripts, particularly when written. Written test cases are usually collected into test suites.

Typical written test case format

A test case is usually a single step, or occasionally a sequence of steps, to test the correct behavior/ functionalities, features of an application. An expected result or expected outcome is usually given. Additional information that may be included:

- test case ID
- test case description
- test step or order of execution number
- related requirement(s)
- depth
- test category
- author
- Additional fields that may be included and completed when the tests are executed:
- pass/fail
- remarks/comments

These steps can be stored in a word processor document, spreadsheet, database or other common repository. In a database system, you may also be able to see past test results and who generated the results and the system configuration used to generate those results. These past results would usually be stored in a separate table. Test suites may also contain

- Test summary

- Configuration

Template:

Project Name:

Test Case Template example

Test Case ID: google_10

Test Designed by: <Name>

Test Priority (Low/Medium/High): Med

Test Designed date: <Date>

Module Name: Google login screen

Test Executed by: <Name>

Test Title: Verify login with valid username and password

Test Execution date: <Date>

Description: Test the Google login page

Pre-conditions: User has valid username and password

Dependencies:

S.no	Test Case	Test Steps	Test Data	Expected Output	Actual Output	Pass/ Fail	Comments?
Tc_001	Verify URL	1. Open Browser 2. Type URL 3. Press Enter	https://www.gmail.com	Page should be loaded	loaded	pass	

Post Condition: User is validated with database and successfully login to account. The account session details are logged in database.

- **Test case ID**: Unique ID for each test case. Follow some convention to indicate types of test. E.g. 'TC_UI_1′ indicating 'user interface test case #1′.
- **Test priority (Low/Medium/High)**: This is useful while test execution. Test priority for business rules and functional test cases can be medium or higher whereas minor user interface cases can be low priority. Test priority should be set by reviewer.
- **Module Name** – Mention name of main module or sub module.
- **Test Designed By**: Name of tester
- **Test Designed Date**: Date when wrote
- **Test Executed By**: Name of tester who executed this test. To be filled after test execution.
- **Test Execution Date**: Date when test executed.
- **Test Title/Name**: Test case title. E.g. verify login page with valid username and password.
- **Test Summary/Description**: Describe test objective in brief.
- **Pre-condition**: Any prerequisite that must be fulfilled before execution of this test case. List all pre-conditions in order to successfully execute this test case.
- **Dependencies**: Mention any dependencies on other test cases or test requirement.
- **Test Steps**: List all test execution steps in detail. Write test steps in the order in which these should be executed. Make sure to provide as much details as you can. Tip – to efficiently manage test case with lesser number of fields use this field to describe test conditions, test data and user roles for running test.
- **Test Data**: Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.
- **Expected Result**: What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on screen.
- **Post-condition**: What should be the state of the system after executing this test case?
- **Actual result**: Actual test result should be filled after test execution. Describe system behavior after test execution.
- **Status (Pass/Fail)**: If actual result is not as per the expected result mark this test as failed. Otherwise update as passed.
- **Notes/Comments/Questions**: To support above fields if there are some special conditions which can't be described in any of the above fields or there are questions related to expected or actual results mention those here.

Add following fields if necessary:

- **Defect ID/Link**: If test status is fail, then include the link to defect log or mention the defect number.
- **Test Type/Keywords**: This field can be used to classify tests based on test types. E.g. functional, usability, business rules etc.
- **Requirements**: Requirements for which this test case is being written. Preferably the exact section number of the requirement doc.
- **Attachments/References**: This field is useful for complex test scenarios. To explain test steps or expected result using a visio diagram as a reference. Provide the link or location to the actual path of the diagram or document.

- **Automation? (Yes/No)**: Whether this test case is automated or not. Useful to track automation status when test cases are automated.

Example:

Project Name:	Google Email
Module Name:	Login
Reference Document:	If any
Created by:	Your Name
Date of creation:	DD-MMM-YY
Date of review:	DD-MMM-YY

TEST CASE ID	TEST SCENARIO	TEST CASE	PRE-CONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	POST CONDITION	ACTUAL RESULT	STATUS (PASS/ FAIL)
TC_LOGIN_001	Verify the login of Gmail	Enter valid User Name and valid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Valid User Name>	Successful login	Gmail inbox is shown		
				2. Enter Password	<Valid Password>				
				3. Click "Login" button					
TC_LOGIN_001	Verify the login of Gmail	Enter valid User Name and invalid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Valid User Name>	A message "The email and password you entered don't match" is shown			
				2. Enter Password	<Invalid Password>				
				3. Click "Login" button					
TC_LOGIN_001	Verify the login of Gmail	Enter invalid User Name and valid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Invalid User Name>	A message "The email and password you entered don't match" is shown			
				2. Enter Password	<Valid Password>				
				3. Click "Login" button					
TC_LOGIN_001	Verify the login of Gmail	Enter invalid User Name and invalid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Invalid User Name>	A message "The email and password you entered don't match" is shown			
				2. Enter Password	<Invalid Password>				
				3. Click "Login" button					

EXERCISE: write detailed Test cases for face book Login and signup pages by using template /*Attach printout here */

EXERCISE: Write detailed Test cases for face book Login and signup pages by using template. Attach printout here.

TEST DESCRIPTION:

Project Name:	**Facebook**
Test-Scenario ID:	**TS-FB-01**
Test Title:	**Verify the Login function of Facebook**
Test Priority:	**medium**
Module Name:	**Facebook Login**
Designed & Executed By:	**Musadique Hussain SE-21031**
Designed & Execution Date:	**06-May-2024**

TEST CASES:

S. No	Test Case	Test Steps	Test Data	Expected Output	Actual Output	Status
TC_001	Verify URL	1. 2. Open Browser 3. Type URL Press Enter	https://www.facebook.com	Login page should be loaded	Loaded	PASS
TC_001	Input Valid Email addrs and Valid Password	1. 2. Enter Valid Email. 3. Enter Valid Password Click 'Login' button	Email = "ValidEmail@gmail.com" Password = "Valid Password"	Login successfully	Logged In Successfully	PASS
TC_002	Input InValid Email addrs and InValid Password	1. 2. Enter Invalid Email. 3. Enter InValid Password Click 'Login' button	Email = "InValidEmail@gmail.com" Password = "Invalid Password"	Login page show invalid credential	Account not Found	PASS

TC_003	Input Valid Email addrs and InValid Password	1. 2. Enter Valid Email. 3. Enter InValid Password Click 'Login' button	Email = "ValidEmail@gmail.com" Password = "Invalid Password"	Should show the invalid password prompt	Invalid Password	PASS
TC_004	Input InValid Email addrs and Valid Password	1. 2. Enter Invalid Email. 3. Enter Valid Password Click 'Login' button	Email = "InValidEmail@gmail.com" Password = "Valid Password"	Account will not access	Not found Email	PASS
TC_005	Checking Case sensitivity: Input Valid Email addrs and Valid Password in Uppercase	1. Enter Valid Email in Uppercase. 2. Enter Valid Password uppercase Click 'Login' button 3.	Email = "ValidEmail@gmail.com" Password = "VALID PASSWORD"	Login successfully	Logged In Successfully	PASS
TC_006	Login through Phone No: Enter valid Phone No and valid Password	1. 2. Enter Valid Phone. 3. Enter Valid Password Click 'Login' button	Phone = valid "033333333" Password = "Valid Password"	Access account	Logged in Successfully	PASS
TC_007	Login through Phone No: Enter Invalid Phone No and Invalid Password	1. 2. Enter Invalid Phone. 3. Enter invalid Password Click 'Login' button	Phone = invalid "023450" Password = "invalid Password"	Login page show invalid credential	Account not Found	PASS
TC_008	Login through Phone No: Enter invalid Phone No and valid Password	1. 2. Enter Invalid Phone. 3. Enter Valid Password Click 'Login' button	Phone = Invalid "0233e" Password = "Valid Password"	Account will not access	Not found Email	PASS
TC_009	Login through Phone No: Enter valid Phone No and invalid Password	1. 2. Enter valid Phone. 3. Enter Invalid Password Click 'Login' button	Phone = valid "02345678910" Password = "invalid Password"	Should show the invalid password prompt	Invalid password	PASS
TC_010	Expanding TC-005 by replacing initial zero with country code	1. Enter Valid country code and Phone no. Enter Valid Password 2. Click 'Login' button 3.	Phone = valid " 92 234567910" Password = "Valid Password"	Login Successfully	Logged in Successfully	PASS
TC_011	Expanding TC-005 by replacing initial zero with invalid country code	1. Enter invalid country code and valid Phone no. Enter Valid Password 2. Click 'Login' button 3.	Phone = invalid " 93 234567910" Password = "Valid Password"	Invalid phone no	Logged in Successfully	FAIL
TC_012	Expanding TC-005 by removing initial zero	1. Enter valid Phone no without initial zero. Enter Valid Password 2. Click 'Login' button 3.	Phone = valid "234567910" Password = "Valid Password"	Login successfully	Logged in Successfully	PASS
TC_013	Login through Username: Enter Username and valid Password	1. 2. Enter Username. 3. Enter valid Password Click 'Login' button	Username = "abcd" Password = "valid Password"	Should not login. *Because mention field required only Email and Mobile*	Logged into Account	FAIL
TC_014	Login through Username: Enter Username and invalid Password	1. 2. Enter Username. 3. Enter invalid Password Click 'Login' button	Username = "abcd" Password = "invalid Password"	Invalid password	Incorrect password	PASS
TC-015	Check "Forget Password" Link	1. Click on "Forget Password"	Nil	Goto to another Page	Open Forget Pass Page	PASS
TC-016	Check "Create New Account" Link	1. Click on "Create New Account"	Nil	Goto to another Page	Open New Account Page	PASS

Software Quality Engineering LAB Manual

TEST DESCRIPTION:

Project Name:	**Facebook**
Test-Scenario ID:	**TS-FB-02**
Test Title:	**Verify the Sign-up function of Facebook**
Test Priority:	**medium**
Module Name:	**Facebook Sign Up**
Designed & Executed By:	**Musadique Hussain SE-21031**
Designed & Execution Date:	**06-May-2024**

TEST CASES:

S.No	Test Case	Test Steps	Test Data	Expected Output	Actual Output	Status
TC-001	Verify Sign Up Form	1. Open Browser 2. Type URL 3. Press Enter	https://www.facebook.com/r.php?r=101\	Login page should be loaded	Loaded	PASS
TC-002	Proceeding form without Entering Data	1. Click the Sign-Up button	Nil	Sign-Up Fail	Invalid Credential	PASS
TC-003	Check Name fields by Numeric value	1. Entering number in First Name field. 2. Entering number in Second Name Field. 3. Entering valid Data to other fields 4. Click on Sign Up	F. Name = '123' S. Name = '456'	Numeric Data would not be allowed	Enter Valid Name which are commonly Used	PASS
TC-004	Check Name fields by Special character	1. Entering Special Char in First Name field. Entering Char in Second Name Field. 2. Entering valid Data to other fields 3. Click on Sign Up 4.	F. Name = '@#' S. Name = '$%'	Special Char would not be allowed	Enter Valid Name which are commonly Used	PASS
TC-005	Check Name field by valid Name	1. Entering Special Char in First Name field. Entering Char in Second Name Field. 2. Entering valid Data to other fields 3. Click on Sign Up 4.	F. Name = 'Muhammad' S. Name = 'Kabeer'	Account created Successfully	Sign-Up successful	PASS
TC-006	Checking Email field by entering Address without '@' tag	1. Enter Email address without @ tag. 2. Entering valid Data to other fields 3. Click on Sign Up	Email = "Abcd.gmail.com"	Email not Accepted	Invalid Email	PASS
TC-007	Checking Email field by entering Address with '@' tag	1. Enter Email address with @ tag. 2. Entering valid Data to other fields 3. Click on Sign Up	Email = "Abcd@gmail.com"	Email Accepted	Sign-Up successfully	PASS
TC-008	Verify the Phone No Field by entering Less than 10 number	1. Enter the phone No less than 10. 2. Entering valid Data to other fields 3. Click on Sign Up	Phone = "23456789"	Not Allowed	Proceeded further	FAIL
TC-009	Verify the Phone No Field by entering the Number without initial zero.	1. Enter the Valid phone No Without initial zero. Entering valid Data to other fields 2. Click on Sign Up 3.	Phone = "1234567890"	Allowed	Proceeded further	PASS
TC-010	Verify the Phone No Field by entering the Number with initial zero.	1. Enter the Valid phone No With initial zero. Entering valid Data to other fields 2. Click on Sign Up 3.	Phone = "02345678090"	Allowed	Proceeded further	PASS
TC-011	Verify the Phone No Field by Replacing initial zero with country code	1. Enter the Valid phone and stuffing country code. Entering valid Data to other fields 2. Click on Sign Up 3.	Phone = "92-2345678900"	Allowed	Proceeded further	PASS
TC-012	Verify the valid Phone No Field by Replacing initial zero with invalid country code	1. Enter the Valid phone No and stuffing invalid country code. Entering valid Data to other fields 2. Click on Sign Up 3.	Phone = "93-2345678900"	Not Allowed	Proceeded further	FAIL
TC-013	Checking Password Field by less than 5 Character	1. Enter 4 character in password field. 2. Entering valid Data to other fields 3. Click on Sign Up	Password = "ABCD"	Not Allowed as it unsafe	Enter greater than 5 characters	PASS
TC-014	Checking Password Field by greater than 5 Character	1. Enter 6 character in password field. 2. Entering valid Data to other fields 3. Click on Sign Up	Password = "ABC123"	Allowed	Proceeded Further	PASS
TC-015	Checking Date of Birth Field for less than Five-year-old person	1. Set date for 4-year-old or 2017 birth person. Entering valid Data to other fields 2. Click on Sign Up 3.	Month = jun, Date = 01, Year = 2017	Should not accepted as it specified in SRS	Invalid Date	PASS
TC-016	Checking Date of Birth Field for greater than Four-year-old person	1. Set date for 5-year-old or 2016 birth person. Entering valid Data to other fields. 2. Click on Sign Up 3.	Month = jun, Date = 01, Year = 2016	Should accepted as it specified in SRS	Proceeded Further	PASS
TC-017	Checking Date of Birth Field for less than 117-year-old person	1. Set date for 116-year-old or 1905 birth person. Entering valid Data to other fields. 2. Click on Sign Up 3.	Month = jun, Date = 01, Year = 1905	Should accepted as it specified in SRS	Proceeded Further	PASS

TC-018	Checking Date of Birth Field for greater than 116-year-old person	1. Set date for 117-year-old or 1904 birth person. Entering valid Data to other fields. 2. Click on Sign Up 3.	Month = jun, Date = 01, Year = 190	Should not accepted as it specified in SRS	Invalid Date	PASS
TC-019	Skip the Selection of Gender radio button	1. Skip selection of Gender. 2. Data to other fields. 3. Click on Sign Up	Nil	Not allow	Select gender	PASS
TC-020	Select Male Gender	1. Selecting Male Radio btn. 2. Data to other fields. 3. Click on Sign Up	Nil	Go further	Proceeded Further	PASS
TC-021	Select Female Gender	1. Selecting Female Radio btn. 2. Data to other fields. 3. //	Nil	Go further	Proceeded Further	PASS

Lab 03: white box Testing

White-box testing is also known as structure testing or glass-box testing. It focuses on the procedural details (or internal structure) of the software when generating test data and test cases. The derived test cases guarantee that all independent paths within a module have been exercised at least once and that all logical decisions have been exercised on their true and false sides. Whitebox testing is performed early in the testing process White-box testing is mostly used at unit level testing.

What do you verify in White Box Testing?

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of white box testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

Working process of white box testing:

- **Input:** Requirements, Functional specifications, design documents, source code.
- **Processing:** Performing risk analysis for guiding through the entire process.
- **Proper test planning:** Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated. • **Output:** Preparing final report of the entire testing process.

How do you perform White Box Testing?

To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

Step 1. Understand the Source Code

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and

prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly. **Step 2. Create Test Cases and Execute**

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools

Example:

Consider the following piece of code

```
Printme (int a, int b) {          ----------  Printme is a function     int
result = a+ b;
  If (result> 0)
        Print ("Positive", result)
  Else
        Print ("Negative", result)
  }                               -----------  End of the source code
```

The goal of WhiteBox testing in software engineering is to verify all the decision branches, loops, statements in the code.To exercise the statements in the above white box testing example, WhiteBox test cases would be

- A = 1, B = 1
- A = -1, B = -3

White Box Testing Techniques

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product. Below are a few coverage analysis techniques a box tester can use:

Statement Coverage: - This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.it ensure every single line of code is tested.

Branch Coverage: - This technique checks every possible path (if-else and other conditional loops) of a software application.it ensure every branch (e.g. true or false) is tested.

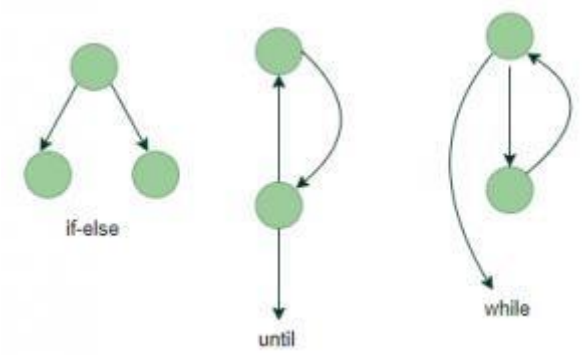
Path Coverage: – Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program. it ensure all possible paths are tested.

Basis/Independent Path Testing: In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

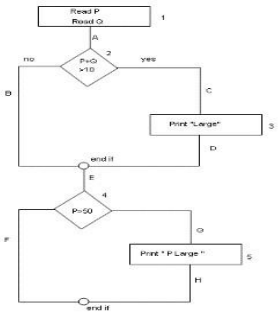
Steps:

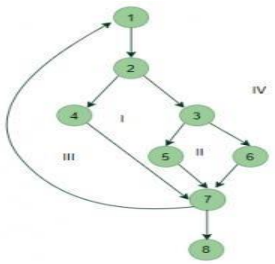
1. Make the corresponding control flow graph
2. Calculate the cyclomatic complexity
3. Find the independent paths
4. Design test cases corresponding to each independent path

Flow graph notation: It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.



Cyclomatic Complexity: It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G, V(G) is its cyclomatic complexity.
Calculating V(G):
V(G) = P + 1, where P is the number of predicate nodes in the flow graph
V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
V(G) = Number of non-overlapping regions in the graph **Example:**





V(G) = 4 (Using any of the above formulae)
No of independent paths = 4
#P1: 1 – 2 – 4 – 7 – 8
#P2: 1 – 2 – 3 – 5 – 7 – 8
#P3: 1 – 2 – 3 – 6 – 7 – 8
#P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8 **Example**
:

- Read P

- Read Q
- IF P+Q > 100 THEN
- Print "Large" • ENDIF
- If P > 50 THEN • Print "P Large"
- ENDIF

Number of test cases required:

❖ **Statement Coverage:** 1A-2C-3D-E-4G-5H = 1

❖ **Branch Coverage:** 1A-2C-3D-E-4G-5H , 1A-2B-E-4F = 2

❖ **Path Coverage:** 1A-2B-E-4F, 1A-2B-E-4G-5H,1A-2C-3D-E-4G-5H, 1A-2C-3D-E-4F=4

Code Coverage Calculation:

Statements Coverage: Simply, these are what you have in the boxes and diamond shapes. You can cover all the statements in the flowchart by writing **1 Test Case** that follows the following route1A-2C-3D-E-4G-5H. (p=60,q=50)

Let us say your test case covers the following route - 1A -> 2B -> E -> 4F. There are **total of 5 Statements** in your flowchart, You **cover only 3 Statements**. Now the Statement Coverage will be - **(3/5)*100**. Your Test Case, therefore, has **60% of Statement Coverage**

Branches/Decisions Coverage: Decisions that you can take in the process flow diagram (For example, if you consider the statement 2, there are two branches to it)You can cover all the branches in the flowchart by writing **2 Test Cases** that follow the following two routes 1A-2C-3D-E-4G-5H, 1A-2B-E-4F .(p=60,Q=50 , p=30,q=30)

Let us say your test case covers the following route - 1A -> 2B -> E -> 4F. There are **total of 4 Decision** in your flowchart (2 decisions for Statement no 2 and 2 decisions for Statement no 4). You **cover only 2 Branches/Decisions**. Now the Branch/Decision Coverage will be- **(2/4)*100**. Your Test Case, therefore, has **50% of Branch Coverage**

Paths Coverage: paths that you can take to travel in the flow chart from the start to the end .You can cover all the paths in the flowchart by writing **4 Test Cases** that follow the following four routes1A-2B-E-4F, 1A-2B-E-4G-5H, 1A-2C-3D-E-4G-5H,1A-2C-3D-E-4F,Therefore, the Path Coverage is 4.Let us say your test case covers the following route - 1A -> 2B -> E -> 4F There are **total of 4 Paths** in your flowchart. You **cover only 1 Path**. Now the Path Coverage will be- **(1/4)*100**. Your Test Case, therefore, has **25% of Path Coverage.**

Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code.

Advantages of White Box Testing

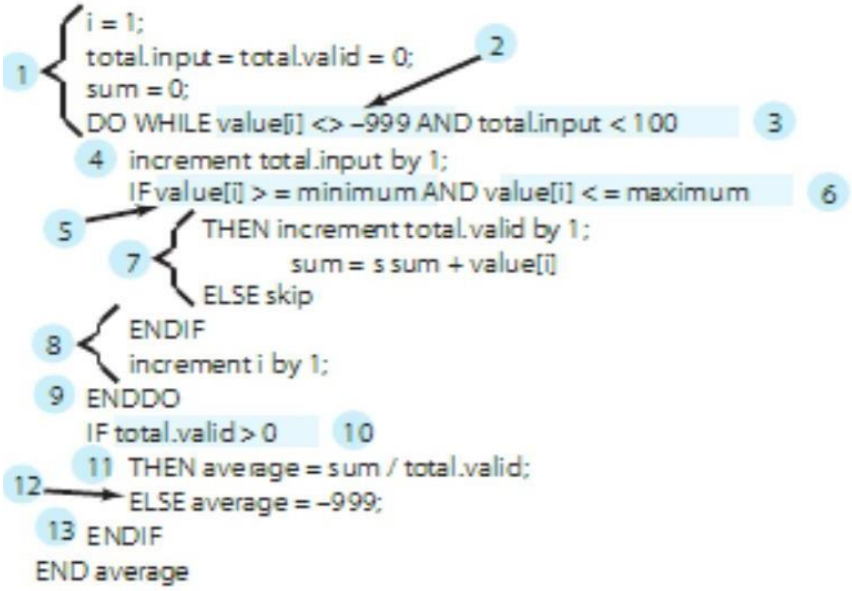
- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Disadvantages of White Box Testing • White box testing can be quite complex and expensive.

- Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
- White box testing requires professional resources, with a detailed understanding of programming and implementation.
- White-box testing is time-consuming, bigger programming applications take the time to test fully.

Exercise:

1. Consider the code given below. Calculate cyclomatic complexity and no. of linearly independent paths and Design Testcases for each path.



To calculate the cyclomatic complexity of the given code, we can use the formula:

V(G) = E - N + 2
Where:
(E) is the number of edges
(N) is the number of nodes

First, let's count the number of nodes and edges in the flowchart representation of the code:

Nodes:
1. Start
2. Initialization
3. DO while loop
4. If statement inside the loop
5. End If
6. Increment i
7. End DO-while
8. If-Else statement after the loop
9. End

Edges:
- Start to Initialization
- Initialization to DO while loop
- DO while loop to If statement inside the loop
- If statement inside the loop to End If
- If statement inside the loop to Increment i
- Increment i to DO while loop
- End If to DO while loop
- DO while loop to If-Else statement after the loop
- If-Else statement after the loop to End

Now, let's count the number of predicate nodes (decision points) in the flow graph:
- There is one predicate node, which is the If statement inside the DO while loop.

Using the formula (V(G) = E - N + 2):
[V(G) = 9 - 7 + 2 = 4]

So, the Cyclomatic complexity (V(G)) is 4.

The number of linearly independent paths is given by (P + 1), where (P) is the number of predicate nodes. In this case, (P = 1), so:

[text{Number of linearly independent paths} = 1 + 1 = 2]

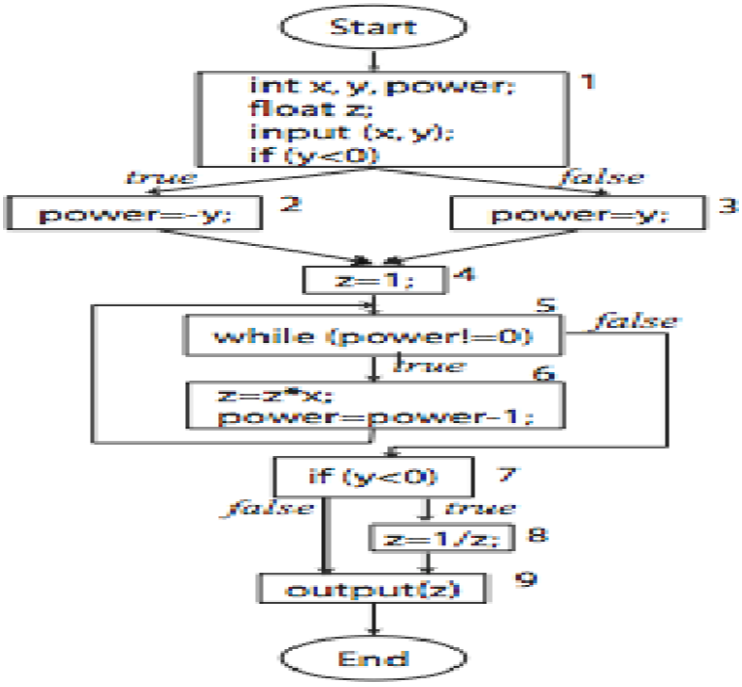
Now, let's design test cases for each path:

Path 1:
Start
Initialization
DO while loop
If statement inside the loop (true)
Increment i
DO while loop
If-Else statement after the loop
End
Test case: Provide input values that satisfy the condition in the If statement inside the loop.

Path 2:
- Start
Initialization
DO while loop
If statement inside the loop (false)
DO while loop
If-Else statement after the loop
End

Test case: Provide input values that do not satisfy the condition in the If statement inside the loop.

2. Consider the flow chart given below and design test cases by using statement coverage method,AND Independent path method also execute test cases using template (provided in lab 2).



Test Cases using Statement Coverage Method:

1. **Test Case ID**: TC1
    - **Test Scenario**: Test when y is negative.
    - **Test Case**: Verify the output when y is negative.
    - **Pre-condition**: None
    - **Test Steps**:
        1. Input x = 2, y = -3
        2. Follow the flowchart.
    - **Test Data**: x = 2, y = -3
    - **Expected Result**: Output z should be 0.125

    . Post-condition: None
2. **Test Case ID**: TC2
    - **Test Scenario**: Test when y is positive.
    - **Test Case**: Verify the output when y is positive.
    - **Pre-condition**: None
    - **Test Steps**:
        1. Input x = 3, y = 4

2. Follow the flowchart.

- **Test Data**: x = 3, y = 4
- **Expected Result**: Output z should be 81.
- **Post-condition**: None

Test Cases using Independent Path Method:

1. **Test Case ID**: TC3
    - **Test Scenario**: Test when y is negative.
    - **Test Case**: Verify the output when y is negative.
    - **Pre-condition**: None
    - **Test Steps**:
        1. Input x = 2, y = -3
        2. Follow the flowchart.
    - **Test Data**: x = 2, y = -3
    - **Expected Result**: Output z should be 0.125
    - **Post-condition**: None
2. **Test Case ID**: TC4
    - **Test Scenario**: Test when y is positive.
    - **Test Case**: Verify the output when y is positive.
    - **Pre-condition**: None
    - **Test Steps**:
        1. Input x = 3, y = 4
        2. Follow the flowchart.
    - **Test Data**: x = 3, y = 4
    - **Expected Result**: Output z should be 81.
    - **Post-condition**: None

Test Case Execution Table

Test case ID	Test Scenario	Test case	Pre - Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status pass/fail

TC1	Negative Y	Check IF Y is Negative	None	1. Input x = 2 y = -3	X = 2 y = -3	Output z should be 0.125	None	Output z = 0.125	Pass
TC2	Positive Y	Check If y is positive.	None	1. Input x = 3, y = 4.	x = 3, y = 4	Output z should be 81	None	Output = 81	Pass
TC3	Negative Y	Check If Y is Negative	None	1. Input x = 2, y = -3. 2. Follow the diagram	1. X = 2, y = -3	Output Z should be 0.125	None	Output z = 0.125	Pass
TC4	Positive Y	Check IF Y is Positive	None	1. Input x = 3, y = 4. 2. Follow the diagram	x = 3, y = 4	Output z should be 81.	None	Output Z = 0.125	Pass

To verify the functionality, follow these procedures outlined in the "Test Steps" column. In the "Actual Results" section, document your observations. Finally, compare the obtained results with the anticipated outcomes in the "Expected Results" column to determine if each test case is successful (Pass) or unsuccessful (Fail).

3.Write a program (Any Language) to find odd or even number. Design and execute test casesusing branch coverage method using template.

Code

```python
def odd_or_even(number):
    if number % 2 == 0:
        return "Even number"
    else:
        return "Odd number"

Test_Cases = [
    (4, "Even number"),
    (3, "Odd number"),
    (-2, "Even number"),
    (-7, "Odd number"),
    (6, "Even number"),
]

for Test_input, expected_output in Test_Cases:
    result = odd_or_even(Test_input)
    print(f"Input: {Test_input}, Expected Output: {expected_output}, Actual Output: {result}")
```

Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\> & C:/Python311/python.exe e:/lab.py
Input: 4, Expected Output: Even number, Actual Output: Even number
Input: 3, Expected Output: Odd number, Actual Output: Odd number
Input: -2, Expected Output: Even number, Actual Output: Even number
Input: -7, Expected Output: Odd number, Actual Output: Odd number
Input: 6, Expected Output: Even number, Actual Output: Even number
PS E:\>
```

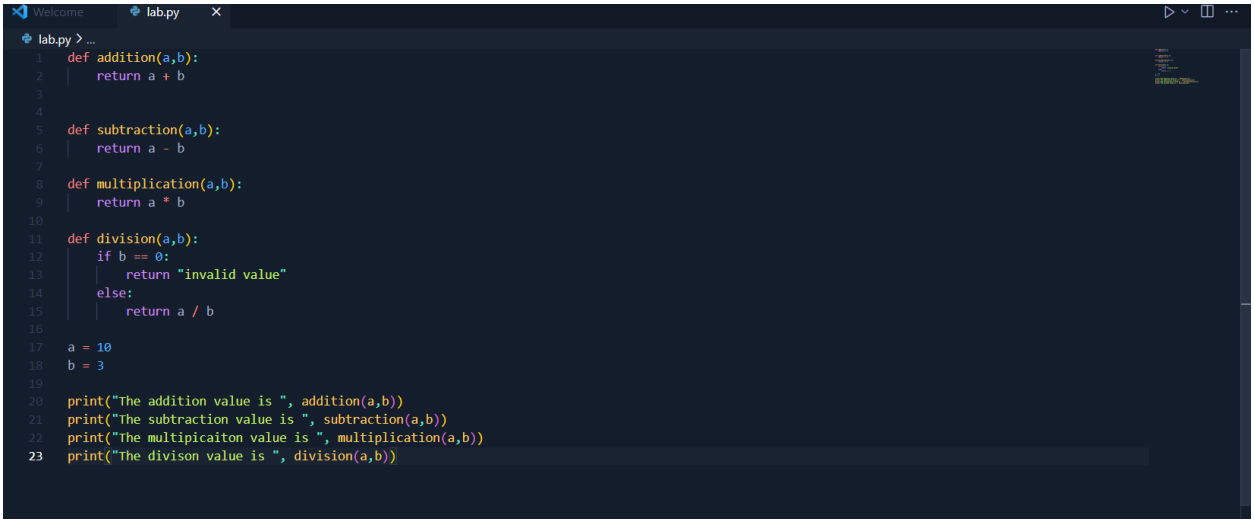
Test case ID	Test Scenario	Test case	Pre - Condition	Test Steps	Test Data	Expected Result	Post Condition	Actual Result	Status pass/fail
TC1	Even Number	Check if the number is even.	None	1. Input number = 2	number = 2	Expected result is "Even"	None	Actual result is "Even"	Pass
TC2	Odd Number	Check if the number is odd.	None	1. Input number = 7	number = 7	Expected result is "Odd"	None	Actual result is "Odd"	Pass
TC3	Negative Even Number	Check if the negative	None	1. Input numb	number = -4	Expected result is "Even"	None	Actual result is	Pass

		even number is detected correctly.		er = -4				"Even"	
TC4	Negative Odd Number	Check if the negative odd number is detected correctly.	None	1. Input number = -9	number = -9	Expected result is "Odd"	None	Actual result is "Odd"	Pass
TC5	Zero	Check if zero is identified as even.	None	1. Input number = 0	number = 0	Expected result is "Even"	None	Actual result is "Even"	Pass

All test cases have been executed successfully. The observed results perfectly align with the anticipated outcomes, resulting in a "Pass" status for each one.

4... Write a program (Any Language) to perform addition, subtraction, multiplication and division. Observe the output and Test it using any one of white box technique .Attached allprintout here.

Code



```python
def addition(a,b):
    return a + b


def subtraction(a,b):
    return a - b

def multiplication(a,b):
    return a * b

def division(a,b):
    if b == 0:
        return "invalid value"
    else:
        return a / b

a = 10
b = 3

print("The addition value is ", addition(a,b))
print("The subtraction value is ", subtraction(a,b))
print("The multipicaiton value is ", multiplication(a,b))
print("The divison value is ", division(a,b))
```

Output

```
PS E:\> & C:/Python311/python.exe e:/lab.py
The addition value is  13
The subtraction value is  7
The multipicaiton value is  30
The divison value is  3.3333333333333335
PS E:\>
```

Description:

Project Name:	Basic-Calculator(say)
Test Suite ID:	Calculate TESE(SE-21031)
Test Title:	Test the basic operators in Calculator
Test Priority:	Medium
Module Name:	Do Calculation
Designed By:	**Musadique Hussain**
Designed Date:	**6th May 6, 2024**
Executed By:	**Musadique Hussain**
Executed Date:	**6th May 2024**
Description of Test:	Verify the addition, subtraction, multiplication and division of two numbers(Also test division by zero)
PREREQUISITES:	
Pre-Condition:	Not required
Dependencies:	No dependencies in this test.

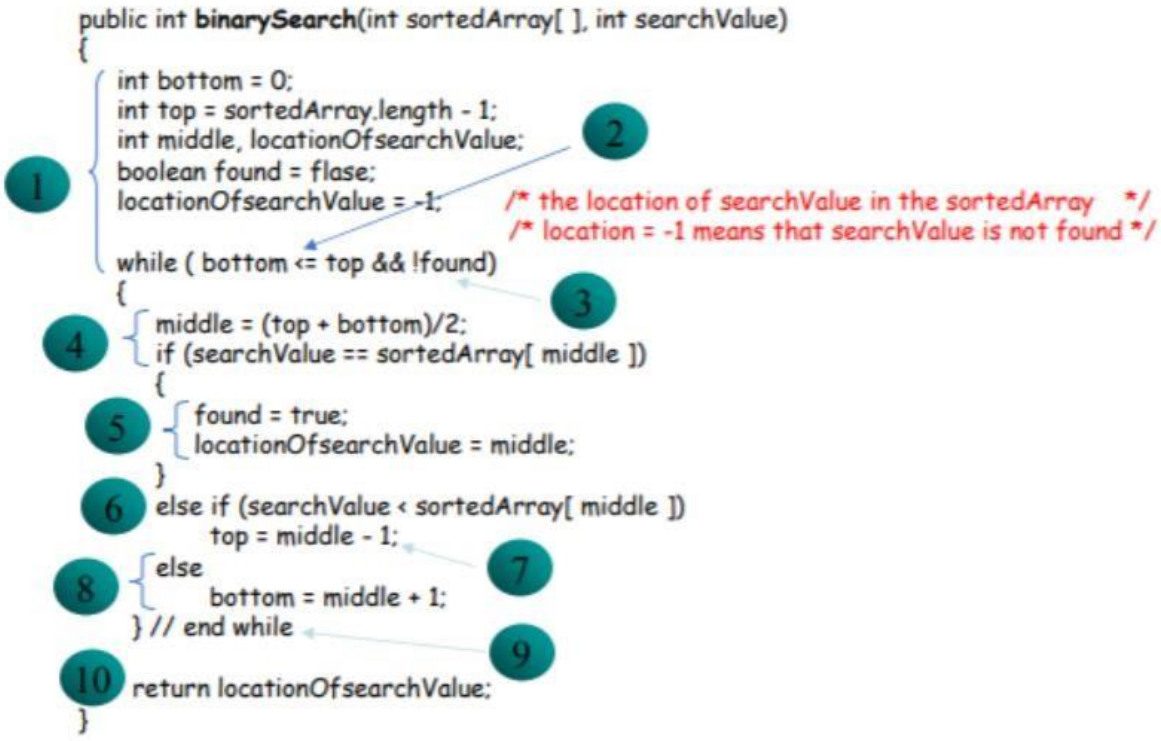
Test Cases:

S. No	Test Case	Test Steps	Test Data	Expected Output	Actual Output	Test case Status Pass/Fail
1	Addition	Perform Addition operation	a = 10, b = 3	13	13	Pass
2	Subtraction	Perform Subtraction operation	a = 10, b = 3	7	7	Pass
3	Multiplication	Perform Multiplication operation	a = 10, b = 3	30	30	Pass
4	Division	Perform Division operation	a = 10, b = 3	3.333333	3.33333	Pass

Lab 04 : White box Path testing for Binary Search and Bubble Sort Program

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Task 01: Recall the logic of Binary Search Tree and apply Any white Box method to test the application. Attach all printout here [Code snippets ,output screenshots , test Summary reports]



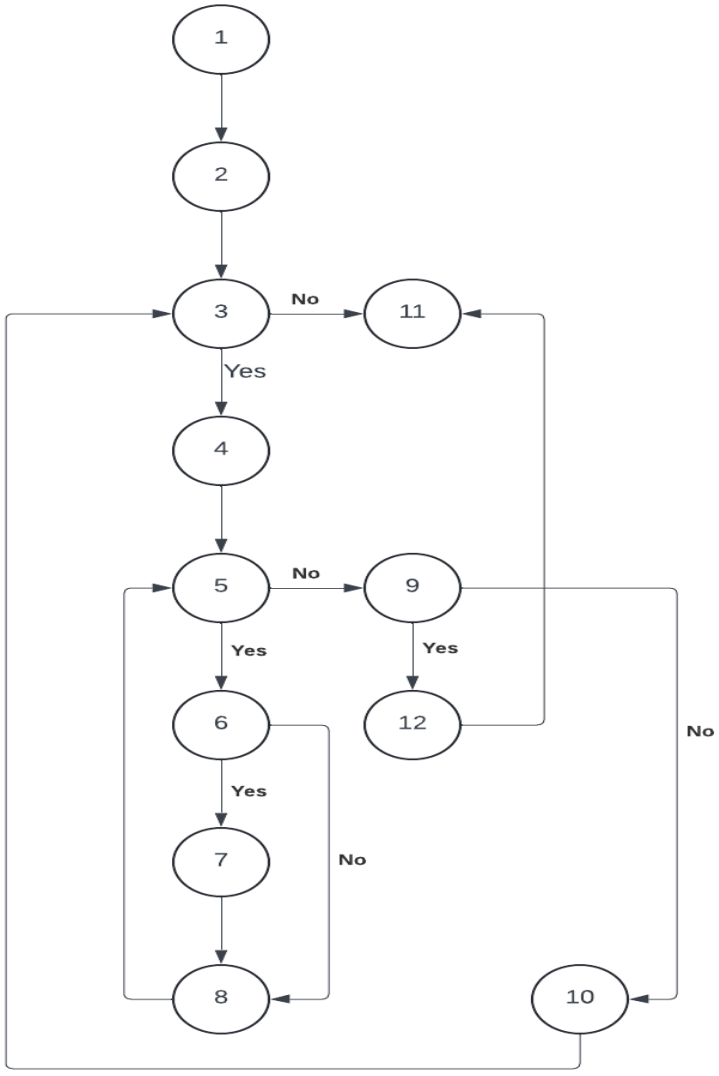
Bubble Sort: This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Task 02: recall the logic of bubble sort algorithm; and apply Any white Box method to test the application. Attach all printout here [Code snippets ,output screenshots , test Summary reports]

Table 1: Part of Program Code

```
1   public static void bubbleSort(int[] array)
2       {
3           int tmp;
4           boolean flag = false;
5           for(int i = array.length-1;i >= 0;i--)
6               {
7                for(int j=0;j<i;j++)
8                   {
9                    if(array[j]>array[j+1])
10                      {
11                        tmp = array[j];
12                        array[j] = array[j+1];
13                        array[j+1] = tmp;
14                        flag = true;
15                      }
16                   }
17              if(!flag)   break;
18              }
19      }
```

Control Flow Graph



CYCLOMATIC COMPLEXITY - V(G)

Since V(G) = E - N + 2

Where;
E = Number of edges = 15
N = Number of nodes = 12

Therefore;
V(G)= 15 - 12 + 2
= 5

NUMBER OF INDEPENDENT PATHS

Following are the independent paths:

P1: 1-2-3-4-5-6-7-8-9-10-11

P2: 1-2-3-11

P3: 1-2-3-4-5-9-10-11

P4: 1-2-3-4-5-9-12-11

P5: -1-2-3-4-5-6-8-9-12-11

Test Cases

Case ID	Description	Input Data	Expected Output	Actual Output	Test Case Status
1	Enter array and verify bubble sort algorithm P1(1,2,3,4,5,6,7,8,9,10,11)	[4,1,3,5,2]	[1,2,3,4,5]	[1,2,3,4,5]	PASS
2	Enter array and verify bubble sort algorithm P2(1,2,3,4,5,6,8,9,12,11)	[1,2,3,5,4], flag value is set to false, so break is executed and the program ends.	[1,2,3,5,4]	[1,2,3,5,4]	PASS
3	Enter array and verify bubble sort algorithm P3(1,2,3,11)	[1,2,3,4,5], this is the step when the whole array has been traversed and sorted, i.e. var has been decremented up to the length of the array	[1,2,3,4,5]	[1,2,3,4,5]	PASS
4	Enter array and verify bubble sort algorithm P4(1,2,3,4,5,9,10,11)	[3,1,2,5,4], This is the case when 1st element is compared with all the other elements present in the array, so here 3, is compared with 1, then,2, then 5, if 3< 5 so next time 5 is checked with 4 and flag value is set to true	[1,2,3,4,5]	[1,2,3,4,5]	PASS
5	Enter array and verify bubble sort algorithm P5(1-2-3-4-5-9-12-11)	[1,2,3,4,5], when j=1 means one-time	[1,2,3,4,5]	[1,2,3,4,5]	PASS

		traversing has done and no flag is set			

Test Case 1

```
Original array: 4 1 3 5 2
Sorted array: 1 2 3 4 5
```

Test Case 2

```
Original array: 1 2 3 5 4
Sorted array: 1 2 3 4 5
```

Test Case 3

```
Original array: 1 2 3 4 5
Sorted array: 1 2 3 4 5
```

Test Case 4

```
Original array: 3 1 2 5 4
Sorted array: 1 2 3 4 5
```

Test Case 5

```
Original array: 1 2 3 4 5
Sorted array: 1 2 3 4 5
```

Lab 05:

Exercise: Select your previous project and Design Test cases for at least 5 components using any of white box strategy also answer the following questions

i. Draw a comparison of selected techniques with all other ones.

White box testing techniques include the following

Statement Coverage: Ensures that every statement in the code is executed at least once.

Branch Coverage: Ensures that every branch (decision) in the code is executed at least once.

Path Coverage: Ensures that all possible paths in the code are executed.

Condition Coverage: Ensures that every condition in a decision takes on all possible outcomes at least once.

Loop Coverage: Ensures that all loops in the code are executed with zero, one, and multiple iterations.

Comparison:

Statement Coverage: Simple and ensures that no line of code is skipped. However, it might miss bugs in complex conditional logic.

Branch Coverage: More thorough than statement coverage as it covers every decision point. It is still possible to miss some logical errors.

Path Coverage: Most comprehensive as it covers all possible execution paths, but it is complex and often infeasible for large applications due to the combinatorial explosion of paths.

Condition Coverage: Focuses on individual conditions within decision points, which can catch errors that branch coverage might miss but is less thorough than path coverage.

Loop Coverage: Specifically targets loops, ensuring they are tested thoroughly, but does not cover other parts of the code as comprehensively as path coverage.

Test Cases of Nibbles Application

1. **User Authentication**

Feature Description: This feature allows users to securely log in to the application using their credentials. It includes input validation, session management, and redirection upon successful login.

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
TC-01	Verify login with valid credentials	1. Navigate to login page. 2. Enter valid username and password. 3. Click login.	Username: user Password: password	User successfully logs in and is redirected	As Expected	Pass
TC-02	Verify login with invalid credentials	1. Navigate to login page 2. Enter invalid username or password 3. Click login.	Username: invalid Password: invalid	Login fails, user receives an error message	As Expected	Pass
TC-03	Verify case sensitivity in login credentials	1. Navigate to login page. 2. Enter valid username	Username: user Password: PASSWORD	Login fails if credentials' case doesn't match	As Expected	Pass
		and incorrect case password. 3.				

		Click login.				
TC-04	Verify user session creation upon successful login	1. Log in with valid credentials. 2. Check if user session is created.	Username: user Password: password	User session is created and maintained	As Expected	Pass
TC-05	Verify redirection to dashboard upon successful login	. Login with valid credentials. 2. Check if user is redirected to dashboard.	Username: user Password: password	User is redirected to the dashboard	As Expected	Pass

2. Meal Selection

Feature Description: This feature allows users to view, filter, add, and remove meals from their selection. It includes dietary preference filters and updates the total cost based on selected meals.

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
TC-01	Verify viewing all available meal options	1. Login. 2. Navigate to meal selection page.	N/A	User successfully logs in and is redirected	As Expected	Pass

TC -02	Verify filtering meals based on dietary preferences	1. Login. 2. Navigate to meal selection page. 3. Apply filter based on	Preference: Vegetarian	Login fails, user receives an error message	As Expected	Pass
		dietary preference.				
TC-03	Verify adding a meal to selection	1. Login. 2. Navigate to meal selection page. 3. Select a meal. 4. Add meal to selection	Meal: Pasta	Login fails if credentials' case doesn't match	As Expected	Pass
TC-04	Verify removing a meal from selection	1. Login. 2. Navigate to meal selection page. 3. Select a meal from current selection. 4. Remove meal from selection.	Meal: Pasta	User session is created and maintained	As Expected	Pass
TC-05	Verify total cost update when meals are added/removed	1. Login. 2. Navigate to meal selection page. 3. Add and remove meals. 4. Check the total cost.	Meal: Pasta Meal: Salad	User is redirected to the dashboard	As Expected	Pass

3. Subscription Management

Feature Description: This feature allows users to manage their subscription plans, including viewing current details, upgrading, downgrading, and ensuring costs reflect changes immediately.

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
TC -01	Verify viewing current subscription details	1. Login. 2. Navigate to subscription management page. Selection page.	N/A	User sees their subscription detail slogs in and is redirected	As Expected	Pass
TC -02	Verify upgrading subscription plan	1. Login. 2. Navigate to subscription management page. 3. Select upgrade plan. 4. Confirm upgrade.	New Plan: Premium	Subscription plan upgrades successfully	As Expected	Pass
TC-03	Verify downgrading subscription plan	1. Login. 2. Navigate to subscription management page. 3. Select downgrade plan. 4. Confirm downgrade.	New Plan: Basic	Subscription plan downgrades successfully	As Expected	Pass
TC-04	Verify subscription cost updat e upon changes	1. Login. 2. Navigate to subscription management page. 3. Change subscription plan. 4. Check updated cost.	Plan Change: Basic to Premium	Subscription cost updates accurately	As Expected	Pass

TC-05	Verify immediate reflection of subscription changes	1. Login. 2. Navigate to subscription management page. 3. Change subscription plan. 4. Verify changes in user's	Plan Change: Basic to Premium	Changes reflect immediately in user's account	As Expected	Pass
		account immediately.				

4. Payment Processing

Feature Description: This feature handles the payment processing for subscriptions and meal purchases. It includes entering payment details, processing payments, and handling success or failure notifications.

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
TC-01	Verify entering payment details	1. Login. 2. Navigate to payment page. 3. Enter payment details. 4. Submit payment.	Card Number: 1234 5678 9123 4567 Expiry: 12/23	User can enter payment details	As Expected	Pass

TC -02	Verify processing payment with valid details	1. Login. 2. Navigate to payment page. 3. Enter valid payment details. 4. Submit payment.	Card Number: 1234 5678 9123 4567 Expiry: 12/23	Payment processes successfully	As Expected	Pass
TC-03	Verify payment decline with invalid details	1. Login. 2. Navigate to payment page. 3. Enter invalid payment details. 4. Submit payment.	Card Number: 0000 0000 0000 0000 Expiry: 00/0	Payment is declined, user receives an error	As Expected	Pass
TC-04	Verify confirmation	1. Login. 2. Navigate to	Card Number: 1234 5678	User receives a	As Expected	Pass
	email upon successful payment	payment page. 3. Enter valid payment details. 4. Submit payment. 5. Check email for confirmation.	9123 4567 Expiry: 12/23	confirmation email		

TC-05	Verify notification of payment failure	1. Login. 2. Navigate to payment page. 3. Enter invalid payment details. 4. Submit payment. 5. Check notification for payment failure.	Card Number: 0000 0000 0000 0000 Expiry: 00/00	User is notified of payment failur	As Expected	Pass

ii. Why you choose this technique?

Path Testing was chosen because it ensures all possible execution paths are tested, which is critical for identifying logic errors and ensuring robustness in the "Nibbles" application. This technique is particularly useful given the application's complexity, involving multiple user actions and decision points.

ii. On what level we can apply white box and when will we should stop software testing

Application Level and Stopping Criteria for White Box Testing Application Level:

Unit Testing: White box testing is most commonly applied at the unit testing level, where individual functions or methods are tested.

Integration Testing: It can also be applied during integration testing to ensure that different modules interact correctly.

System Testing: Less commonly used at the system level due to the complexity and the focus on black box testing at this stage.

Stopping Criteria:

Coverage Goals Met: When all planned tests are executed and coverage goals (e.g., 100% statement, branch, or path coverage) are met.

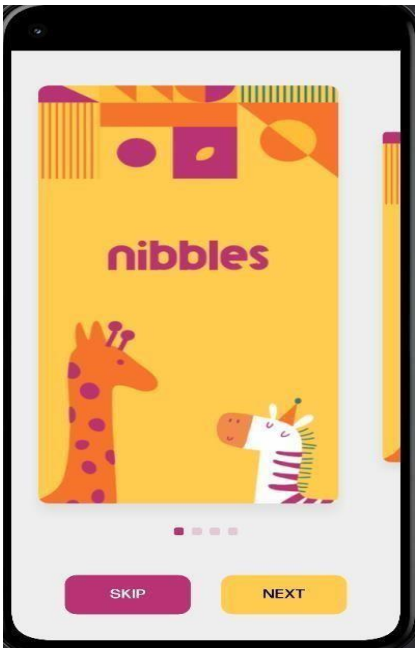
Bug Rates Decrease: When the rate of finding new bugs decreases significantly and remaining bugs are considered minor or low-risk.

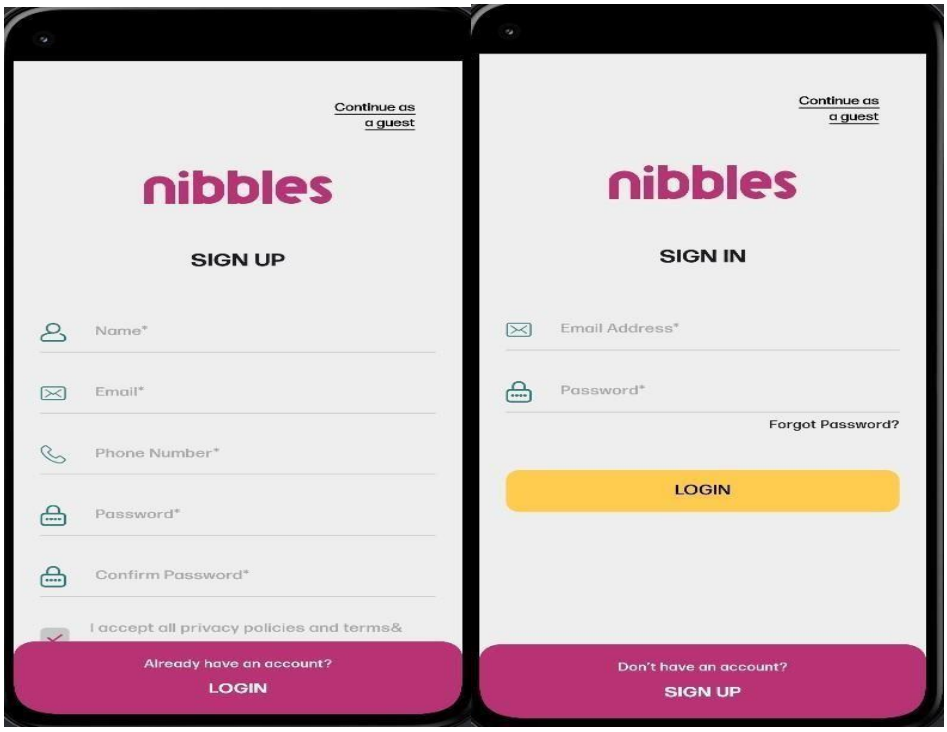
Deadlines/Time Constraints: When the testing phase reaches its scheduled end, balancing thorough testing with project timelines.

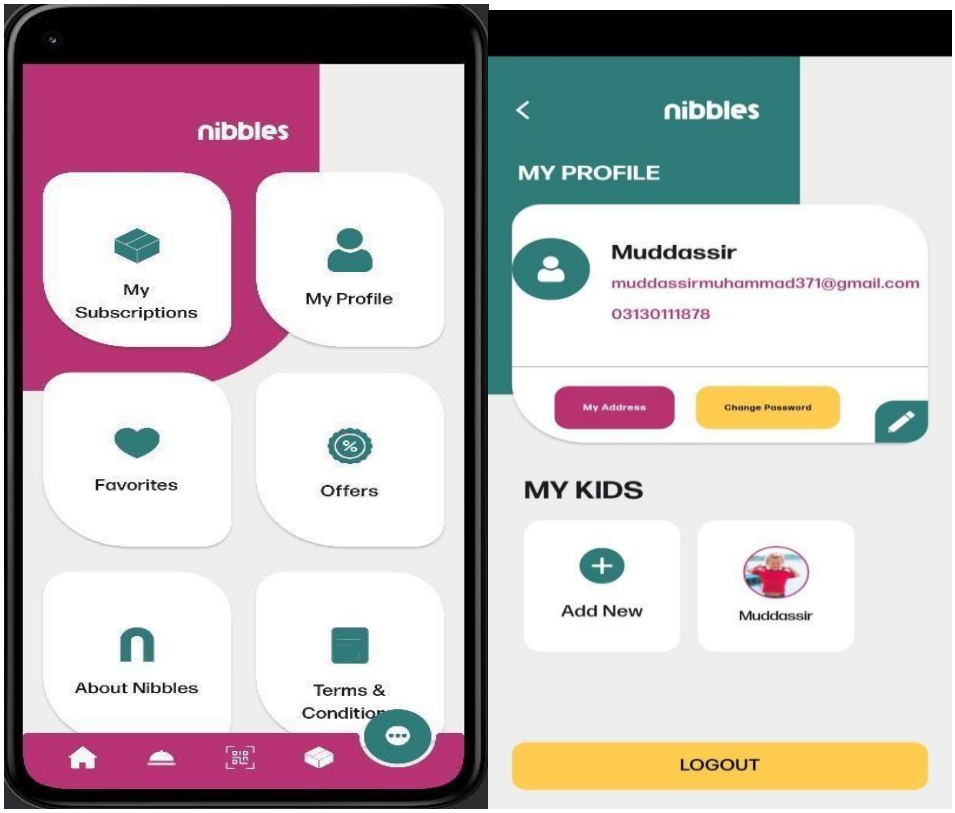
Acceptance Criteria Met: When the software meets the acceptance criteria defined by stakeholders or clients.

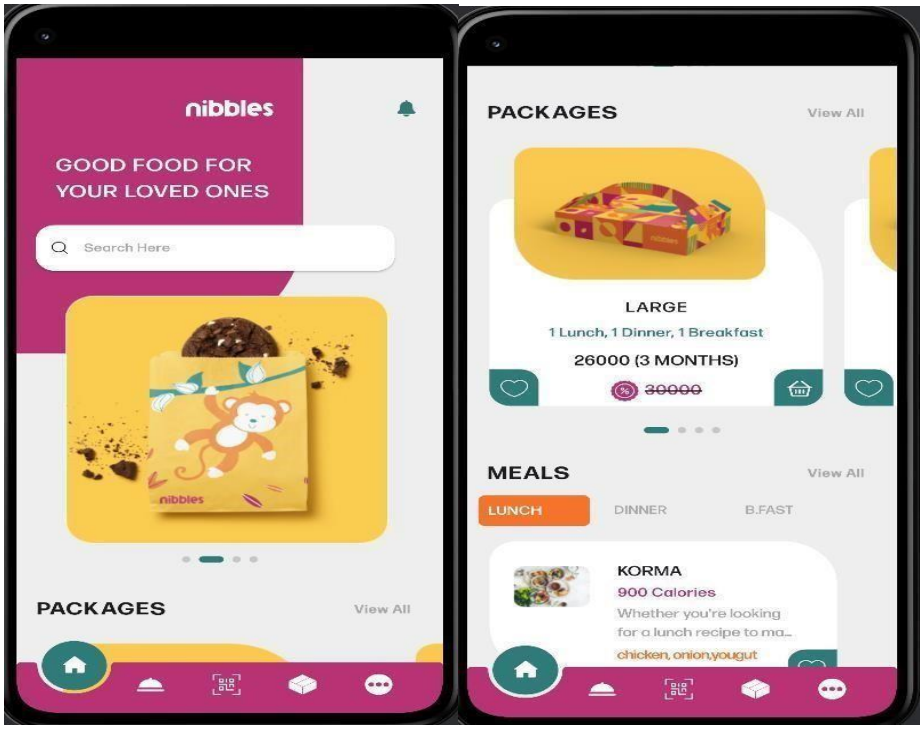
Risk Assessment: When remaining risks are deemed acceptable based on a risk assessment and business decisions.

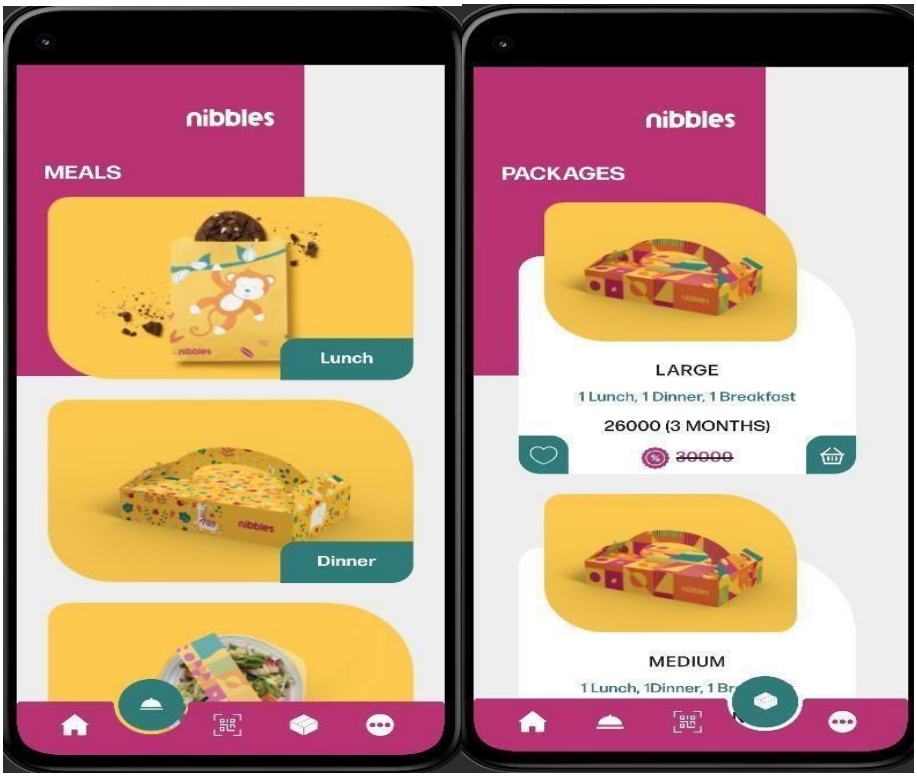
Applications Screenshots




Continue as a guest
nibbles
SIGN UP
Name*
Email*
Phone Number*
Password*
Confirm Password*
I accept all privacy policies and terms&
Already have an account?
LOGIN
Continue as a guest
nibbles
SIGN IN
Email Address*
Password*
Forgot Password?
LOGIN
Don't have an account?
SIGN UP


nibbles
My Subscriptions
My Profile
Favorites
Offers
About Nibbles
Terms &
nibbles
MY PROFILE
Muddassir
muddassirmuhammad371@gmail.com
03130111878
My Address
Change Password
MY KIDS
Add New
Muddassir
LOGOUT


nibbles
GOOD FOOD FOR
YOUR LOVED ONES
Search Here
PACKAGES
View All
PACKAGES
View All
LARGE
1 Lunch, 1 Dinner, 1 Breakfast
26000 (3 MONTHS)
30000
MEALS
View All
LUNCH
DINNER
B.FAST
KORMA
900 Calories
Whether you're looking
for a lunch recipe to ma...
chicken, onion,yougut

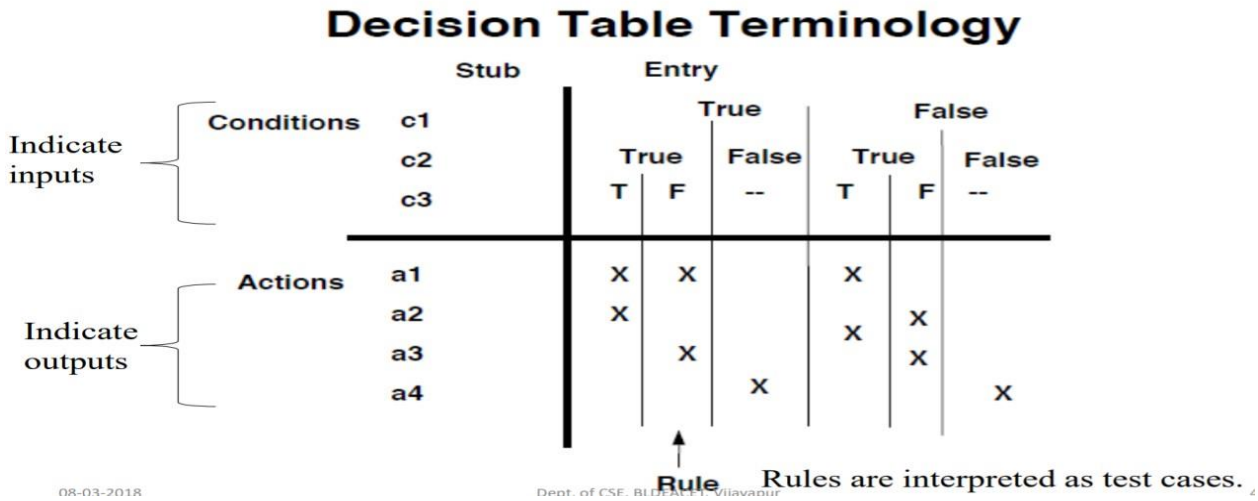

nibbles
MEALS
Lunch
Dinner
nibbles
PACKAGES
LARGE
1 Lunch, 1 Dinner, 1 Breakfast
26000 (3 MONTHS)
30000
MEDIUM

Lab 06: Black Box Testing: Decision Table

The decision table is a software testing technique which is used for testing the system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior are captured in a tabular form. This table helps you deal with different combination inputs with their associated outputs. Also, it is known as the causeeffect table because of an associated logical diagramming technique called cause-effect graphing that is basically used to derive the decision table. It is a black box test design technique use to determine the test scenarios for complex business logic. A decision table has 4 portions:

1. Condition stub:
2. Condition entry
3. Action stubs
4. Action entry

The **condition stub** part lists the individual inputs upon which the decision depends, while the **action stub** part lists the alternative actions that may be taken. The entry parts then show the conditions under which each action is selected. This is done by arranging the **condition entry** part into columns, where each column specifies some condition on each of the input values, and then placing a cross in the same column of the **action entry** part to indicate the particular action to be taken. All the conditions of the column must be satisfied in order for the column to be selected.



Decision Table Terminologies:

- The left most column is the stub portion ● The right is the entry portion
- Condition portion is denoted by c's.
- Action portion is denoted by a's
- A column in entry portion is a rule.
- "don't care" entry :The don't care entry has two interpretation:
  -Condition is irrelevant
  -Condition does not apply Also denoted as "n/a" **Decision**

Table Types:

- Decision tables in which all the conditions are binary(true/false, Yes/No, 0/1) are called Limited Entry Decision Tables. For Limited Entry Decision Tables, if n conditions exist, there must be 2 ^n rules.
- If conditions are allowed to have several values, the resulting tables are called Extended Entry Decision Tables.

Developing Decision Tables

Every decision should be given a name and the logic of the decision table is independent of the sequence in which condition rules are written but the action takes place in the order in which events occur. Wherever possible, duplication of terms and meaning should be avoided and only the standardized language must be used.

The steps of building the concerned tables are given below.

1. Firstly figure out the most essential factors to be considered in making a decision. This will identify the conditions involved in the decision. Only those conditions should be selected which have the potential to either occur or not but partial occurrences are not permissible.
2. Determine the most possible steps that can take place under varying conditions and not just under current condition. This step will identify the actions.
3. Calculate all the possible combinations of conditions. For every N number of conditions there are 2*2*2…. (N times) combinations to be considered.
4. Fill the decision rules in the table. Entries in a decision table are filled as Y/N and action entries are generally marked as "X". For the conditions that are immaterial a hyphen "-" is generally put. Decision table is further simplified by eliminating and consolidating certain rules. Impossible rules are eliminated. There are certain conditions whose values do not affect the decision and always result in the same action. These rules can be consolidated into a single rule.

Example1: scenario for an ATM where a decision table would be of use.

Conditions
Withdrawal Amount <= Balance?
Credit Granted?
Actions
Withdrawal granted

A customer requests a cash withdrawal. One of the business rules for the ATM is that the ATM machine pays out the amount if the customer has sufficient funds in their account or if the customer has the credit granted. Already, this simple example of a business rule is quite complicated to describe in text. A decision table makes the same requirements clearer to understand:

- **Step 1 – Analyze the requirement and create the first column**

Requirement: "Withdrawal is granted if requested amount is covered by the balance or if the customer is granted credit to cover the withdrawal amount".

Conditions	Rule1	Rule2	Rule3	Rule4
Withdrawal Amount <= Balance?	T	T	F	F
Credit Granted?	T	F	T	F
Action count	1	1	1	1
Withdrawal granted				

Step 2 – Add column

The number of columns depends on the number of conditions and the number of alternatives for each condition. If there are two conditions and each condition can be either true or false, you need 4 columns. If there are three conditions there will be 8 columns and so on.

Mathematically, the number of columns is $2^{\text{conditions}}$. In this case $2^2 = 4$ columns. Now is the time to fill in the T (TRUE) and F (FALSE) for the conditions. The simplest is to say that it should look like this:

Step 3: Determine actions

Conditions	Rule1	Rule2	Rule3	Rule4
Withdrawal Amount <= Balance?	T	T	F	F
Credit Granted?	T	F	T	F
Action count	1	1	1	1
Withdrawal granted	T	T	T	F

Enter actions for each column in the table. You will be able to find this information in the requirement. Enter actions for each column in the table. You will be able to find this information in the requirement. Name the columns (the rules). They may be named R1/Rule 1, R2/Rule 2 and so on, but you can also give them more descriptive names.

Step 4 – Reduce the table

Mark insignificant values with "-". If the requested amount is less than or equal to the account balance it does not matter if credit is granted. In the next step, you can delete the columns that have become identical.
Check for invalid combinations. Invalid combinations are those that cannot happen, for example, that someone is both an infant and senior. Mark them somehow, e.g. with "X". In this example, there are no invalid combinations. Finish by removing duplicate columns. In this case, the first and third column are equal, therefore one of them is removed.

Conditions	**Rule1**	**Rule2**	**Rule3**	**Rule4**
Withdrawal Amount <= Balance?	T	T	F	F
Credit Granted?	T	F	T	F
Action count	1	1	1	1
Withdrawal granted	T	T	T	F

Conditions	**Rule1,2**	**Rule3**	**Rule4**
Withdrawal Amount <= Balance?	T	F	F
Credit Granted?	-	T	F
Action count	2	1	1
Withdrawal granted	T	F	F

Test Case ID	**Balance**	**Withdrawal Amount**	**Credit Granted**	**Expected Results**
1	200	200	Yes	withdrawal granted.
2	100	200	Yes	withdrawal granted.

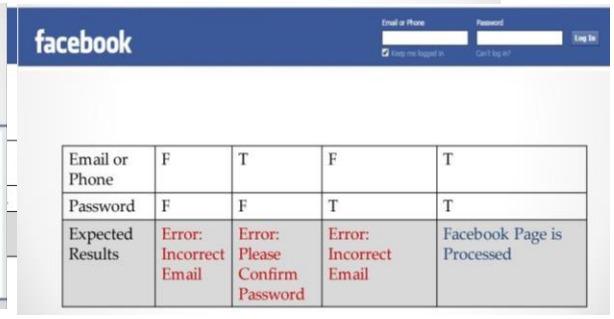
3	100	200	No	Withdrawal denied

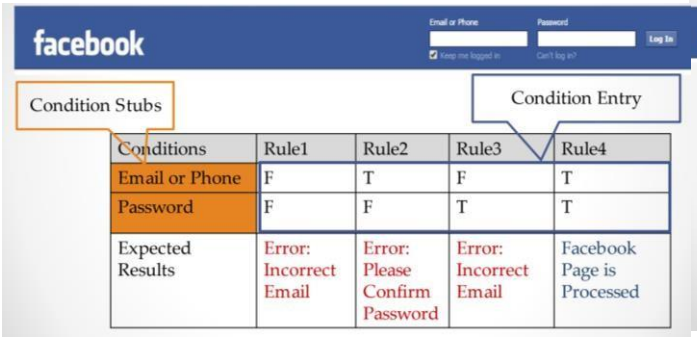
Step 5 – Write Test cases

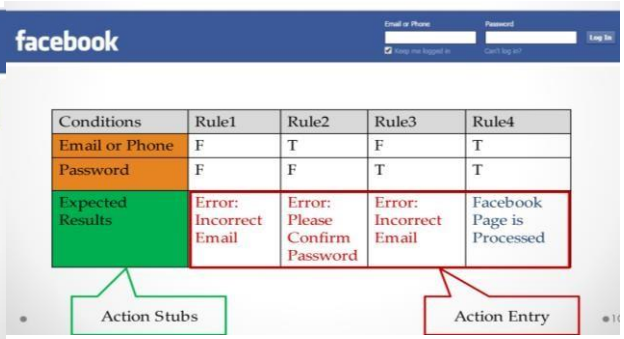
Write test cases based on the table. At least one test case per column gives full coverage of all business rules.

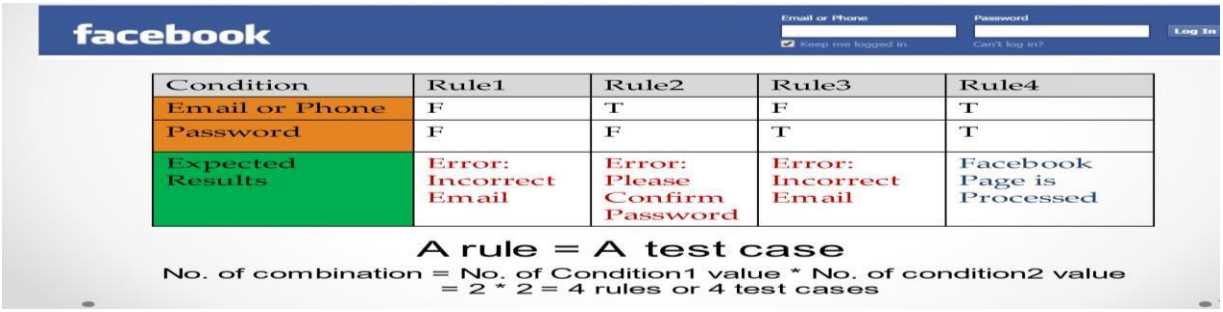
- Test case for R1: balance = 200, requested withdrawal = 200. Expected result: withdrawal granted.
- Test case for R2: balance = 100, requested withdrawal = 200, credit granted. Expected result: withdrawal granted.
- Test case for R3: balance = 100, requested withdrawal = 200, no credit. Expected Result: withdrawal denied.

Example3: face book login









Why is Decision Table Important?

A decision table is an outstanding technique used for testing and requirements management. Some of the reasons why the decision table is important include:

- Decision tables are very much helpful in test design technique.
- It helps testers to search the effects of combinations of different inputs and other software states that implement business rules.
- It provides a regular way of stating complex business rules which benefits the developers as well as the testers.
- It assists in the development process with the developer to do a better job. Testing with all combination might be impractical.
- It the most preferable choice for testing and requirements management.
- It is a structured exercise to prepare requirements when dealing with complex business rules.
- It is also used in model complicated logic.

EXERCISE:

1. Consider Triangle problem : /* Design and develop a program in a language of your choice to solve the triangle problem defined as follows : Accept three integers which are supposed to be the three sides of triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Derive test cases for your program based on decision-table approach, execute the test cases and discuss the results */ and attached all printouts /screenshots/summary report here

Conditions:
- a < b + c?
- b < a + c?
- c < a + b?
- a = b?
- a = c?
- b = c?

a, b, c, form a triangle?

Actions:
- Not a Triangle
- Scalene
- Isosceles
- Equilateral
- Impossible

(i) Make initial decision table consisting of columns (using the formula 2^conditions) Total Columns:

=2^condition

=2^6

=64

Initial Decision Table

	Conditions	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25	c26	c27	c28	c29	c30	c31	c32	c33	c34	c35	c36	c37	c38	c39	c40	c41	c42	c43	c44	c45	c46	c47	c48	c49	c50	c51	c52	c53	c54	c55	c56	c57	c58	c59	c60	c61	c62	c63	c64
c1	a<b+c	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
c2	b<a+c	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
c3	c<a+c	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
c4	a=b	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
c5	a=c	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
c6	b=c	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
	Actions																																																																									
a1	Not triangle									Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
a2	Scalene								Y																																																																		
a3	Equilateral	Y																																																																									
a4	Isosceles			Y		Y	Y																																																																			
a5	Impossible		y	Y		Y																																																																			

(ii) Reduced the table to remove redundant/impossible scenarios.

Reduced Table

	Conditions	c1	c2	c3	c4	c5	c6	c7	c8	c9-c16	c17-c32	c33-c64
c1	a<b+c	T	T	T	T	T	T	T	T	T	T	F
c2	b<a+c	T	T	T	T	T	T	T	T	T	F	0
c3	c<a+c	T	T	T	T	T	T	T	T	F	0	0
c4	a=b	T	T	T	T	F	F	F	F	0	0	0
c5	a=c	T	T	F	F	T	T	F	F	0	0	0
c6	b=c	T	F	T	F	T	F	T	F	0	0	0
	Actions											
a1	Not triangle									Y	Y	Y
a2	Scalene								Y			
a3	Equilateral	Y										
a4	Isosceles				Y		Y	Y				
a5	Impossible		Y	Y		Y						

iii. and (iv) write test cases for reduced decision table and execute code for triangle problem and execute test cases

DESCRIPTION	
Test Title	Verifying the triangle and type of triangle
Test Case ID	TP-001
Test Priority	Low

Test Cases

Sno	Case Description	Inputs			Expected Result	Actual Output	Status
		a	b	c			
Test1	Side a is greater than sum of b and c	7	2	4	Not a valid triangle	Not a valid triangle	Pass
Test2	Side b is greater than sum of a and c	2	8	4	Not a valid triangle	Not a valid triangle	Pass
Test3	Side c is greater than sum of b and a	3	9	3	Not a valid triangle	Not a valid triangle	Pass
Test4	The three sides a b c are equal	5	5	5	Equilateral triangle	Equilateral triangle	Pass
Test5	Two sides are equal	3	3	2	Isosceles triangle	Isosceles triangle	Pass
Test6	All sides are different	5	4	6	Scalene triangle	Scalene triangle	Pass

Designed by	Musadique Hussain
Designed Date	04-06-2024
Executed by	Musadique Hussain
Execution date	11-06-2022
PREREQUISTES	
Pre-Condition	Not required
Dependencies	No dependencies in this test

Code

```
a = int(input("Enter the value of a"))
b = int(input("Enter the value of b"))
c = int(input("Enter the value of c"))

if((a<b+c) & (b<a+c) & (c<b+a)):

    if(a==b):
        if(a==c):
            print("It is an equilateral triangle")
            quit()

            print("It is an isosceles triangle")
        elif(b==c):
            print("It is an isosceles traingle")
        else:
            print("It is a scelene triangle")
else:
    print("Its not a triangle")
```

Output

```
PS E:\> & C:/Python311/python.exe e:/SQE.py
Enter the value of a 7
Enter the value of b 2
Enter the value of c 3
Its not a triangle
```

```
PS E:\> & C:/Python311/python.exe e:/SQE.py
Enter the value of a 6
Enter the value of b 6
Enter the value of c 6
It is an equilateral triangle
```

```
PS E:\> & C:/Python311/python.exe e:/SQE.py
Enter the value of a: 5
Enter the value of b: 3
Enter the value of c: 6
It is a scalene triangle
```

2. Consider a library management system(LMS) which issues book(s) to a registered user by checking the outstanding fee i.e. if a registered user have no pending fee then requested book may be issued(if and only if the potential user has under borrow limit)
i. Make initial decision table consisting of columns (using the formula 2^conditions) ii. Reduced the table to remove redundant/impossible scenarios. iii. write test cases for reduced decision table

Library management system

Conditions:

C1: The User must be registered in library management system

C2: There is no pending fees remaining of the user.

C3: The Potential user is under borrow limit.

Actions:

A1: Book is issued

(i) Make initial decision table consisting of columns (using the formula 2^conditions) Total Columns:

=2^condition

=2^3

=8

Original Decision Table

Condition	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
C1	F	F	F	F	T	T	T	T
C2	F	F	T	T	F	F	T	T
C3	F	T	F	T	F	T	F	T
Actions								
A1	N	N	N	N	N	N	N	Y

(ii) ii. Reduced the table to remove redundant/impossible scenarios.

Reduced Decision Table

Condition	Rule1-4	Rule5-6	Rule 7	Rule 8
C1	F	T	T	T
C2	0	F	F	T
C3	0	0	F	T
Actions				
A1	N	N	N	Y

(iii) and (iv) write test cases for reduced decision table and iv. execute code for LMS and execute test cases

DESCRIPTION	
Test Title	Verifying the LMS users no of books and storage capacity
Test Case ID	TP-001
Test Priority	Low
Designed by	Musadique Hussain
Designed Date	04-06-2024
Executed by	Musadique Hussain
Execution date	11-06-2022
PREREQUISTES	
Pre-Condition	Not required
Dependencies	No dependencies in this test

Test Cases						
Sno	Case Conditions			Expected Result	Actual Result	Status
	c1	c2	c3			
1	F	T	F	User is not registered	User is not registered	Pass
2	T	F	T	Fees is pending	Fees is pending	Pass
3	T	T	F	Borrow limit has already reached	Borrow limit has already reached	Pass
4	T	T	T	Successfully borrowed	Successfully borrowed	Pass

Code

```python
print("Welcome to the Library Management System for registered users")

# List of users and their current number of borrowed books
registered_users = {"James": 2, "Sarah": 4, "Roger": 3}
max_borrow_limit = 4
pending_fees_users = ["James"]

def borrow_book(user, book):
    if user not in registered_users:
        print("\nUser is not registered")
    elif user in pending_fees_users:
        print("\nUser has pending fees")
    elif registered_users[user] >= max_borrow_limit:
        print("\nUser has reached the borrow limit")
    else:
        print("Book successfully borrowed:", book)
        registered_users[user] += 1

# Get user input for borrowing a book
username = input("Enter the username: ")
book_name = input("Enter the book name: ")

borrow_book(username, book_name)
```

Output

```
PS E:\> & C:/Python311/python.exe e:/SQE.py
Welcome to the Library Management System for registered users
Enter the username: Ali
Enter the book name: Alice in wonderland
```

```
PS E:\> & C:/Python311/python.exe e:/SQE.py
Welcome to the Library Management System for registered users
Enter the username: Sarah
Enter the book name: The three musketeers

User has reached the borrow limit
```

```
Welcome to the Library Management System for registered users
Enter the username: James
Enter the book name: Winnie the Pooh

User has pending fees
```

```
Welcome to the Library Management System for registered users
Enter the username: Roger
Enter the book name: Adventure of the Arabian sea
Book successfully borrowed: Adventure of the Arabian sea
PS F:\>
```

Lab 07A: Equivalence Class Partitioning(ECP)

Equivalence partitioning is a software testing technique that divides the input data of a software unit into partition of data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed.

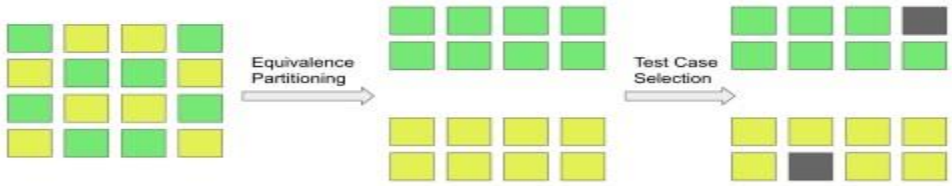
Test cases are designed for equivalence data class. The equivalence partitions are frequently derived from the requirements specification for input data that influence the processing of the test object. A use of this method reduces the time necessary for testing software using less and effective test cases.It can be used at any level of software for testing and is preferably a good technique to use first. In this technique, only one condition to be tested from each partition. Because we assume that, all the conditions in one partition behave in the same manner by the software. In a partition, if one condition works other will definitely work. Likewise we assume that, if one of the condition does not work then none of the conditions in that partition will work.

- Valid Input Class = Keeps all valid inputs.
- Invalid Input Class = Keeps all Invalid inputs.

An input has certain ranges which are valid and other ranges which are invalid. Invalid data here does not mean that the data is incorrect; it means that this data lies outside of specific partition. This may be best explained by the example of a function which takes a parameter "month". The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition. In this example there are two further partitions of invalid ranges. The first invalid partition would be <= 0 and the second invalid partition would be >= 13.

Steps:

- Divide the input domain into classes of data for which test cases can be generated.
- Attempting to uncover classes of errors.
- Derives test cases based on these partitions
- An equivalence class is a set of valid or invalid states of input • Test case design is based on equivalence classes for an input domain. **ECP process:**



Example1:

- A text field permits only numeric characters
- Length must be 6-10 characters long

Partition according to the requirement should be like this:

- While evaluating Equivalence partitioning, values in all partitions are equivalent that's why 05 are equivalent, 6 – 10 are equivalent and 11- 14 are equivalent.
- At the time of testing, test 4 and 12 as invalid values and 7 as valid one.
- It is easy to test input ranges 6–10 but harder to test input ranges 2-600. Testing will be easy in the case of lesser test cases but you should be very careful. Assuming, valid input is 7. That means, you belief that the developer coded the correct valid range (6-10).

Example2:

Check age is between 18 and 60

Invalid	Valid	Invalid
< 18	18 to 60	> 18

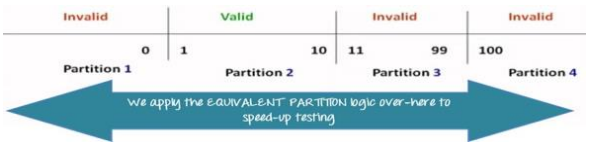
Example3:

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

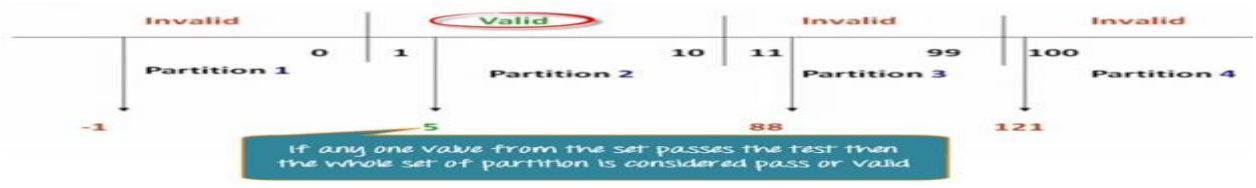
Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid 4. Any 3 Digit Number say -100 is invalid.

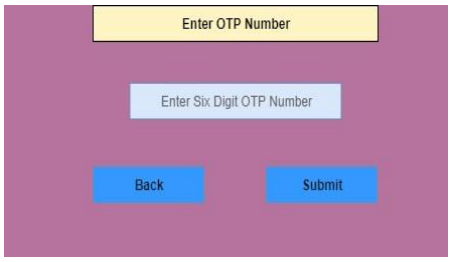
We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.



The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.



Example4: OTP Number = 6 digits



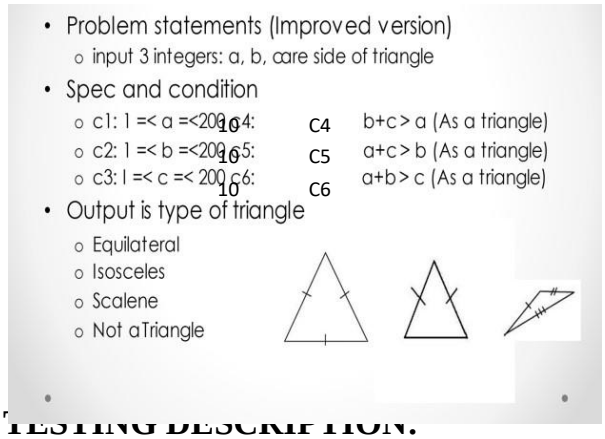
INVALID	INVALID	VALID	VALID
1 Test case	2 Test case	3 Test case	
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

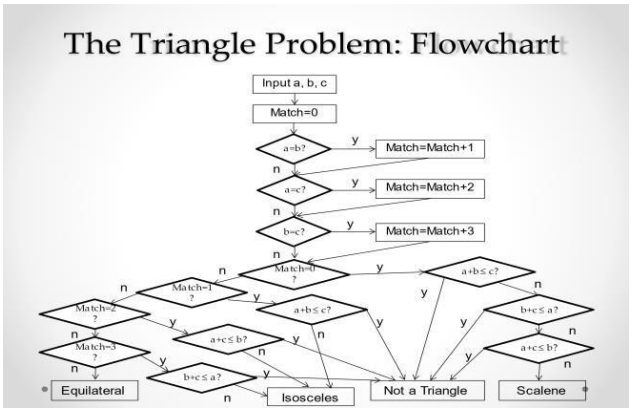
Advantages	Disadvantages
It is process-oriented	All necessary inputs may not cover.
We can achieve the Minimum test coverage	This technique will not consider the condition for boundary value analysis.
It helps to decrease the general test execution time and also reduce the set of test data.	The test engineer might assume that the output for all data set is right, which leads to the problem during the testing process.

EXERCISE: Consider Triangle problem : /* Design and develop a program in a language of your choice to solve the triangle problem defined as follows : Accept three integers which are supposed to be the three sides of triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Derive test cases for your program based on decision-table approach, execute the test cases and discuss the results */

Test Case Name :Equivalence class Analysis for triangle problem
Test Data : Enter the 3 Integer Value(a , b And c)
Pre-Condition : $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$

Brief Description : Check whether given value for a Equilateral, Isosceles , Scalene triangle or can't form a triangle





TESTING DESCRIPTION:

Project Name:	Triangle Problem Decision (Equivalence Partitioning)
Test ID:	Solve_Triangle_EP_TESEXX
Test Title:	Use Equivalence partitioning approach for triangle
	problem
Test Priority:	Low
Module Name:	Identify_Triangle
Test Data:	Enter the 3 Integer Value (a, b and c)
Designed By:	Musadique Hussain
Designed Date:	04/06/2024
Executed By:	Musadique Hussain
Executed Date:	11/06/2024
Description of Test:	Check whether the given values form an equilateral,

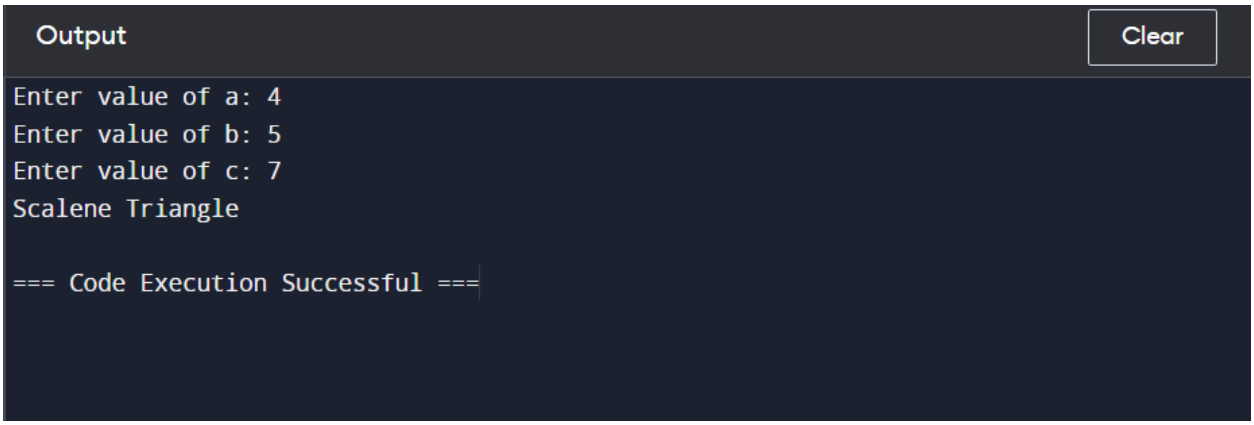
	isosceles, scalene triangle or no triangle using equivalence
	Partitioning approach.

PREREQUISITES:	
Pre-Conditions:	Range in which testing is to be done: 1 <= a <= 10 1<= b <= 10 and 1 <= c <= 10 and a + b + c b < a + c and c < a + b
Dependencies	No dependencies in this test

Code

```
a = int(input("Enter value of a: "))
b = int(input("Enter value of b: "))
c = int(input("Enter value of c: "))

# Checking if the inputs are valid
if a < 1 or a > 10 or b < 1 or b > 10 or c < 1 or c > 10:
    print("Invalid input")
else:
    # Checking if it forms a triangle
    if a + b > c and a + c > b and b + c > a:
        # Checking for equilateral triangle
        if a == b == c:
            print("Equilateral Triangle")
        # Checking for isosceles triangle
        elif a == b or b == c or a == c:
            print("Isosceles Triangle")
        # If not equilateral or isosceles, then it is a scalene triangle
        else:
            print("Scalene Triangle")
    else:
        print("Not a triangle")
```

Output



Test Cases

S.no	Case Description	Input Data			Expected Output	Actual Output	Status	Comments
		a	b	c				
1	Correct Values of a,b,c and all values are equal	5	5	5	Equilateral	Equilateral	Pass	Working Fine
2	Correct Values of a,b,c and all are different	2	5	4	Scalene	Scalene	Pass	Working Fine
3	Correct Values of a,b,c and a & b are equal	4	4	3	Isosceles	Isosceles	Pass	Working Fine
4	Incorrect a and correct values of b and c	0	4	7	Invalid Input	Invalid Input	Pass	Value of a does not lie between 1 and 10 so invalid output
5	Incorrect a and correct values of b and c	11	4	7	Invalid Input	Invalid Input	Pass	Value of a does not lie between 1 and 10 so invalid output
6	Incorrect b and correct values of a and c	3	-1	4	Invalid Input	Invalid Input	Pass	Value of b does not lie between 1 and 10 so invalid output
7	Incorrect b and correct values of a and c	2	12	3	Invalid Input	Invalid Input	Pass	Value of b does not lie between 1 and 10 so invalid output

8	Incorrect c and correct values of a and b	3	8	0	Invalid Input	Invalid Input	Pass	Value of c does not lie between 1 and 10 so invalid output
9	Incorrect c and correct values of a and b	3	7	15	Invalid Input	Invalid Input	Pass	Value of c does not lie between 1 and 10 so invalid output

Test Execution Summary:

Total no. of test cases	**No. of test cases executed**	**No. of test cases passed**	**No. of test cases failed**	**Test Cases Aborted**
9	9	9	0	0

Lab No. 7B: Write test cases Using Boundary Value Analysis

Boundary value analysis is software testing design technique in which tests are designed to include representatives of boundary values. Values on the edge of an equivalence partition or at the smallest value on either side of an edge are taken for testing. The values could be either input or output ranges of a software component. Since these boundaries are common locations for errors that result in software faults they are frequently exercised in test cases. It is a Black Box Testing Method, complements to Equivalence partition and BVA leads to a selection of test cases that exercise bounding values.

The expected input and output values should be extracted from the component specification. The input and output values to the software component are then grouped into sets with identifiable boundaries. Each set, or partition, contains values that are expected to be processed by the component in the same way.

For example, if the input values were months of the year expressed as integers, the input parameter 'month' might have the following partitions:

... -2 -1 0 1 .....12 13 14 15 .....

----------|----------|---------- invalid

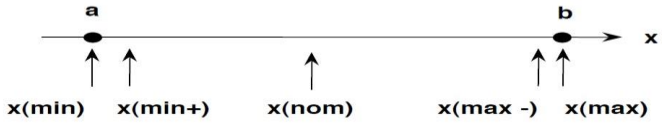
partition 1 valid partition invalid partition 2

The boundaries are the values on and around the beginning and end of a partition. If possible test cases should be created to generate inputs or outputs that will fall on and to either side of each boundary. This would result in three cases per boundary. The test cases on each side of a boundary should be in the smallest increment possible for the component under test. In the example above there are boundary values at 0,1,2 and 11,12,13.

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in normal boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



Example:

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.

- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Example : Input Box should accept the Number 1 to 10;Here we will see the Boundary Value Test Cases

Test Scenario Description	Expected Outcome
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept
Boundary Value = 9	System should accept
Boundary Value = 10	System should accept
Boundary Value = 11	System should NOT accept

Why Boundary Analysis Testing is important?

1. This testing is used to reduce a very large number of test cases to manageable chunks.
2. Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
3. Appropriate for calculation-intensive applications with a large number of variables/inputs

Disadvantages of boundary value analysis

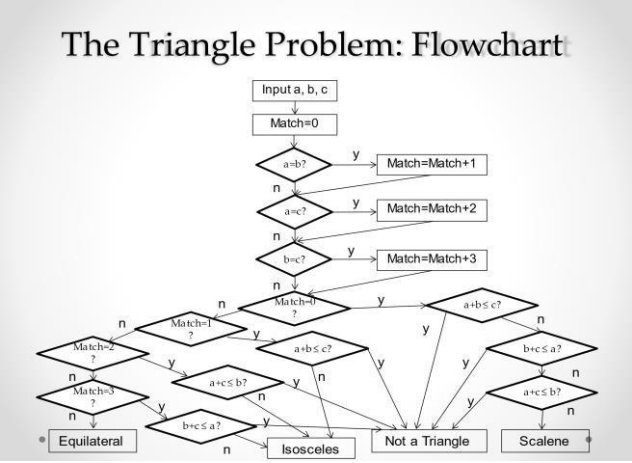
1. The success of the testing using this technique depends on the equivalence classes identified, which further depends on the expertise of the tester and his knowledge of the application. Hence, incorrect identification of equivalence classes leads to incorrect boundary value testing.

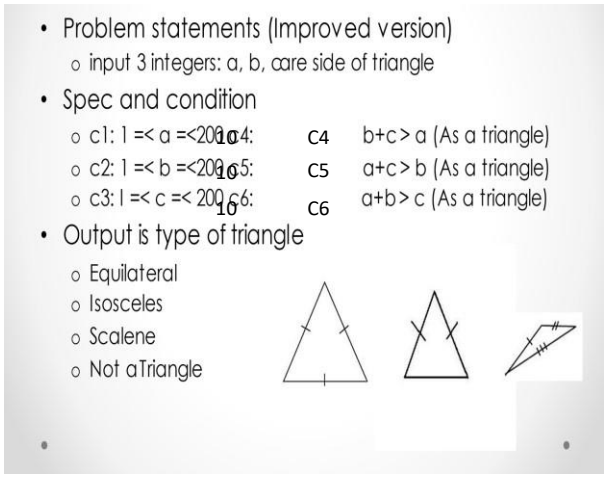
2. Applications with open boundaries or applications not having one-dimensional boundaries are not suitable for this technique. In those cases, other black-box techniques are used.

using BVA method using template

Project Name:	Triangle Problem Decision (Bound Analysis Approach)
Test Case ID:	Solve_Triangle_TESEXX(XX=Rollno.)
Test Title:	Use bound value analysis approach for triangle problem
Test Priority:	Low
Module Name:	Identify_Triangle
Test Data:	Enter the 3 Integer Value (a, b and c)
Designed By:	Musadique Hussain
Designed Date:	04/06/2024
Executed By:	Musadique Hussain
Executed Date:	11/06/2024
Description of Test:	Check whether the given values form an equilateral,

	isosceles, scalene triangle or no triangle using bound
	value analysis approach.
Pre-Conditions:	Range in which testing is to be done: $1 \leq a \leq 10$, $1 \leq b \leq$
	10 and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$
Dependencies:	No dependencies in this test.





EXERCISE: recall the triangle problem and write test cases using BVA method using template

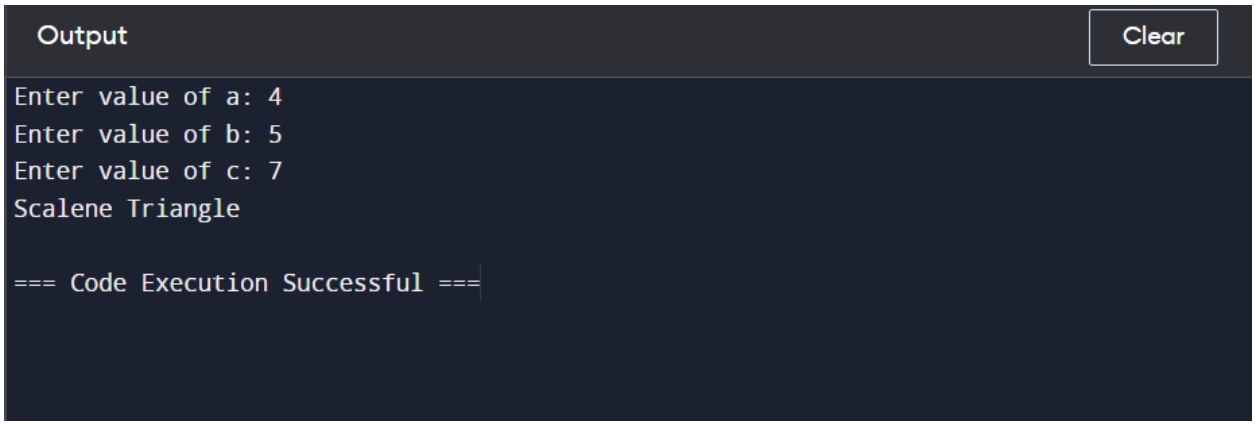
Code

```
def classify_triangle(a, b, c):
    # Check for valid input range
    if not (1 <= a <= 10 and 1 <= b <= 10 and 1 <= c <= 10):
        return "Invalid Input"

    # Check for triangle inequality
    if not (a < b + c and b < a + c and c < a + b):
        return "Not a Triangle"

    # Determine the type of triangle
    if a == b == c:
        return "Equilateral Triangle"
    elif a == b or b == c or a == c:
        return "Isosceles Triangle"
    else:
```

```
        return "Scalene Triangle"

# Test cases using Equivalence Class Partitioning (ECP)
def run_ecp_tests():
    test_cases = [
        (1, 1, 1, "Equilateral Triangle"),
        (2, 2, 3, "Isosceles Triangle"),
        (3, 4, 5, "Scalene Triangle"),
        (1, 2, 3, "Not a Triangle"),
        (0, 5, 5, "Invalid Input"),
        (11, 10, 10, "Invalid Input"),
        (2, 2, 4, "Isosceles Triangle"),
        (5, 5, 1, "Isosceles Triangle"),
        (6, 6, 7, "Isosceles Triangle"),
        (8, 9, 10, "Scalene Triangle")
    ]

    for a, b, c, expected in test_cases:
        result = classify_triangle(a, b, c)
        print(f"Test case {a}, {b}, {c} -> Expected: {expected}, Got: {result}")
        assert result == expected, f"Failed test case {a}, {b}, {c}"

# Test cases using Boundary Value Analysis (BVA)
def run_bva_tests():
    test_cases = [
        (1, 1, 1, "Equilateral Triangle"),
        (1, 10, 10, "Isosceles Triangle"),
        (2, 9, 10, "Scalene Triangle"),
        (10, 10, 10, "Equilateral Triangle"),
        (1, 2, 10, "Not a Triangle"),
        (10, 1, 1, "Not a Triangle"),
        (9, 1, 10, "Scalene Triangle"),
        (2, 2, 3, "Isosceles Triangle"),
        (5, 5, 10, "Not a Triangle"),
        (3, 3, 6, "Isosceles Triangle")
    ]

    for a, b, c, expected in test_cases:
        result = classify_triangle(a, b, c)
        print(f"Test case {a}, {b}, {c} -> Expected: {expected}, Got: {result}")
        assert result == expected, f"Failed test case {a}, {b}, {c}"

if _name_ == "_main_":
```

```
print("Running ECP tests:")
run_ecp_tests()

print("\nRunning BVA tests:")
run_bva_tests()
```

Output



Test Cases

S.no	Case Description	Input Data			Expected Output	Actual Output	Status	Comments
		a	b	c				
1	Boundary Values of a with correct values of b & c	1	1	1	Equilateral	Equilateral	Pass	Working Fine
2	Boundary Values of a with correct values of b & c	0	5	4	Invalid Input	Invalid Input	Pass	Value of a does not lie between 1 and 10 so invalid output
3	Boundary Values of a with correct values of b &	10	4	3	Scalene	Scalene	Pass	Working Fine
4	Boundary Values of a with correct values of b &	11	5	7	Invalid Input	Invalid Input	Pass	Value of a does not lie between 1 and 10 so invalid output

5	Boundary Values of a with correct values of b &	5	4	5	Isosceles	Isosceles	Pass	Working Fine
6	Boundary Values of b with correct values of a & c	3	0	4	Invalid Input	Invalid Input	Pass	Value of b does not lie between 1 and 10 so invalid output
7	Boundary Values of b with correct values of a & c	2	1	3	Scalene	Scalene	Pass	Working Fine
8	Boundary Values of b with correct values of a & c	3	11	2	Invalid Input	Invalid Input	Pass	Value of c does not lie between 1 and 10 so invalid output
9	Boundary Values of b with correct values of a & c	3	10	1	Scalene	Scalene	Pass	Working Fine
10	Boundary Values of b with correct values of a & c	4	5	5	Isosceles	Isosceles	Pass	Working Fine
11	Boundary Values of c with correct values of a & b	3	4	0	Invalid Input	Invalid Input	Pass	Value of c does not lie between 1 and 10 so invalid output
12	Boundary Values of c with correct values of a & b	1	4	1	Isosceles	Isosceles	Pass	Working Fine
13	Boundary Values of c with correct values of a & b	10	10	10	Equilatera l	Equilateral	Pass	Working Fine
14	Boundary Values of c with correct values of a & b	2	6	11	Invalid Input	Invalid Input	Pass	Value of c does not lie between 1 and 10 so invalid output

15	Boundary Values of c with correct values of a & b	3	1	5	Scalene	Scalene	Pass	Working Fine

Test Execution Summary:

Total no. of test cases	**No. of test cases executed**	**No. of test cases passed**	**No. of test cases failed**	**Test Cases Aborted**
15	15	15	0	0

Lab 08:

Exercise: Select your previous project and Design Test cases for at least 5 features using any of black box strategy [you can choose different techniques to test different features]also answer the following questions

Select any (previous) project /Application and test it using Any method of your choice
Q1 Write down the brief details of your project and Draw a comparison of selected techniques with all other ones.

Project Name: Bank Management System

Description: The project is a GUI application in Python where users can register their accounts, log in, and perform banking operations such as deposits, withdrawals, and balance checks. The system is connected to a Microsoft Access database using the pyodbc module to manage and store data.

Comparison of selected technique with all other ones:

1. State-based Testing

Definition: State-based testing focuses on the different states a system can be in and how it transitions between these states based on various inputs.

Advantages:
- Suitable for systems with clear state transitions.
- Effective for capturing and validating different input conditions.
- Can handle both positive and negative input scenarios.
- Useful for systems with a finite number of states and transitions.

Disadvantages:
- Can be complex to model for systems with numerous states and transitions.
- Requires a thorough understanding of the system's state diagram.

2. Black-box Testing

Definition: Testing without knowledge of the internal implementation of the system. It focuses on inputs and expected outputs.

Advantages:
- Tester doesn't need to know the internal workings of the system.
- Can be applied at all levels of software testing (unit, integration, system).

- Useful for validating functionality according to requirements.

Disadvantages:
- Limited coverage of internal states and paths.
- Ineffective for finding issues related to internal states and logic.
- Can miss boundary conditions and special cases if not carefully designed.

3. White-box Testing

Definition: Testing with knowledge of the internal implementation of the system. It focuses on internal paths, conditions, and logic.

Advantages:
- Thorough coverage of internal logic, paths, and conditions.
- Useful for optimizing code and finding hidden errors.
- Can ensure that all paths and branches are tested.

Disadvantages:
- Requires detailed knowledge of the internal code.
- Can be time-consuming and complex for large systems.
- Not suitable for validating external behaviors and interactions.

Q2 What is the purpose of choosing a selected technique/Method?

State-based testing was chosen as it can validate the system's output against different input conditions. The system's behavior can be recorded for positive and negative inputs. Also, such a method is best suited because we have finite testing conditions in our case.

Q3 On what level we can apply black box and why? & Prepare Test cases for at least any 4 main features using any technique. /*Attach Printout here (Also attach screenshots of your Application*/

1) Registration:

The registration feature allows new users to create an account in the Bank Management System. This involves filling out a form with required details such as name, address, contact information, and initial deposit. The information is then stored in the system's database.

Test summary report:

Project Name: Bank Management System
Module Name: Registration
Created By: Musadique Hussain
Date: 16/6/24

Test Scenario ID: REG-001
Test Scenario Description: Verify user can register an account

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
REG - 01-01	Check the UI	Open the application and navigate to registration form	N/A	All elements should be properly aligned	As expected	Pass
REG - 01-02	Register a new user	Click register, fill form, submit	Valid user info	User registered and data stored	As expected	Pass
REG - 01-03	Register with missing fields	Click register, fill form with missing data, submit	Incomplete info	Show error for missing fields	As expected	Pass
REG - 01-04	Register with invalid data	Click register, fill form with invalid data, submit	Invalid info	Show error for invalid data	As expected	Pass
REG - 01-05	Register duplicate user	Click register, fill form with data of an existing user, submit	Duplicate user info	Show error for duplicate user	As expected	Pass
REG - 01-06	Verify data storage	Register a user and check database	Valid user info	User data stored correctly in database	As expected	Pass
REG - 01-07	Register with boundary values	Click register, fill form with boundary values (e.g., max length of text	Boundary values	User registered and data stored	As expected	Pass

		fields), submit				

2) Feature: Login

The login feature enables registered users to access their accounts by entering their username and password. This ensures that only authenticated users can access their personal banking information.

Test summary report:

Project Name: Bank Management System
Module Name: Login
Created By: Musadique Hussain
Date: 16/6/24
Test Scenario ID: LOG-001
Test Scenario Description: Verify user can log in with valid credentials

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
LOG - 01-01	Check the GUI	Open the application and navigate to login form	N/A	All elements should be properly aligned	As expected	Pass
LOG - 01-02	Login with valid credentials	Enter valid username and password, submit	Valid credentials	User logged in and dashboard displayed	As expected	Pass
LOG - 01-03	Login with invalid credentials	Enter invalid username and password, submit	Invalid Credentials	Show error message	As expected	Pass
LOG - 01-04	Login with missing fields	Leave username or password	N/A	Show error for missing fields	As expected	Pass

		empty, submit				
LOG - 01-05	Login with SQL injection	Enter SQL injection code in username or password, submit	SQL injection code	Show error and prevent login	As expected	Pass
LOG - 01-06	Session management	Login, then check if session is maintained	Valid credentials	Session should be maintained correctly	As expected	Pass

3) Feature: Deposit

The deposit feature allows users to add money to their account. This feature provides an interface for users to specify the amount they wish to deposit and updates their account balance accordingly.

Test summary report:

Project Name: Bank Management System
Module Name: Deposit
Created By: Musadique Hussain
Date: 16/6/24
Test Scenario ID: WIT-001
Test Scenario Description: Verify user can deposit money

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
DEP - 01-01	Check the GUI	Open deposit section	N/A	All elements should be properly aligned	As expected	Pass
DEP - 01-02	Deposit valid amount	Enter valid amount, submit	Valid amount	Amount deposit and balance updated	As expected	Pass
DEP - 01-03	Deposit invalid amount	Enter invalid amount (e.g., negative,	Invalid amount	Show error message	As expected	Pass

		zero, more than balance), submit				
DEP - 01-04	Deposit without amount	Leave amount field empty, submit	N/A	Show error for missing fields	As expected	Pass
DEP - 01-05	Verify balance update	Deposit amount and check updated balance	Valid amount	Balance updated correctly in database	As expected	Pass
DEP - 01-06	Deposit boundary values	Enter minimum and maximum allowed withdraw amounts, submit	Maximum values	Amount deposit and balance updated	As expected	Pass
DEP-01-07	Deposit special characters	Enter special characters in amount field, submit	Special characters	Show error message	As expected	Pass

4) Feature: Withdraw

The withdraw feature allows users to take money out of their account. This involves specifying the amount to be withdrawn, which is then deducted from their account balance if sufficient funds are available.

Project Name: Bank Management System
Module Name: Withdraw
Created By: Musadique Hussain
Date: 16/6/24
Test Scenario ID: WIT-001
Test Scenario Description: Verify user can withdraw money

Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
WIT - 01-01	Check the GUI	Open withdraw section	N/A	All elements should be properly aligned	As expected	Pass
WIT - 01-02	Withdraw valid amount	Enter valid amount, submit	Valid amount	Amount withdrawn and balance updated	As expected	Pass
WIT - 01-03	Withdraw invalid amount	Enter invalid amount (e.g., negative, zero, more than balance), submit	Invalid amount	Show error message	As expected	Pass
WIT - 01-04	Withdraw without amount	Leave amount field empty, submit	N/A	Show error for missing fields	As expected	Pass
WIT - 01-05	Verify balance update	Withdraw amount and check updated balance	Valid amount	Balance updated correctly in database	As expected	Pass
WIT - 01-06	Withdraw boundary values	Enter minimum and maximum allowed withdraw amounts, submit	Boundary values	Amount withdrawn and balance updated	As expected	Pass
WIT-01-07	Withdraw special characters	Enter special characters in amount field, submit	Special characters	Show error message	As expected	Pass

5) Feature: Balance Check

The balance check feature allows users to view their current account balance. This feature is available on the dashboard and provides an easy way for users to keep track of their finances.

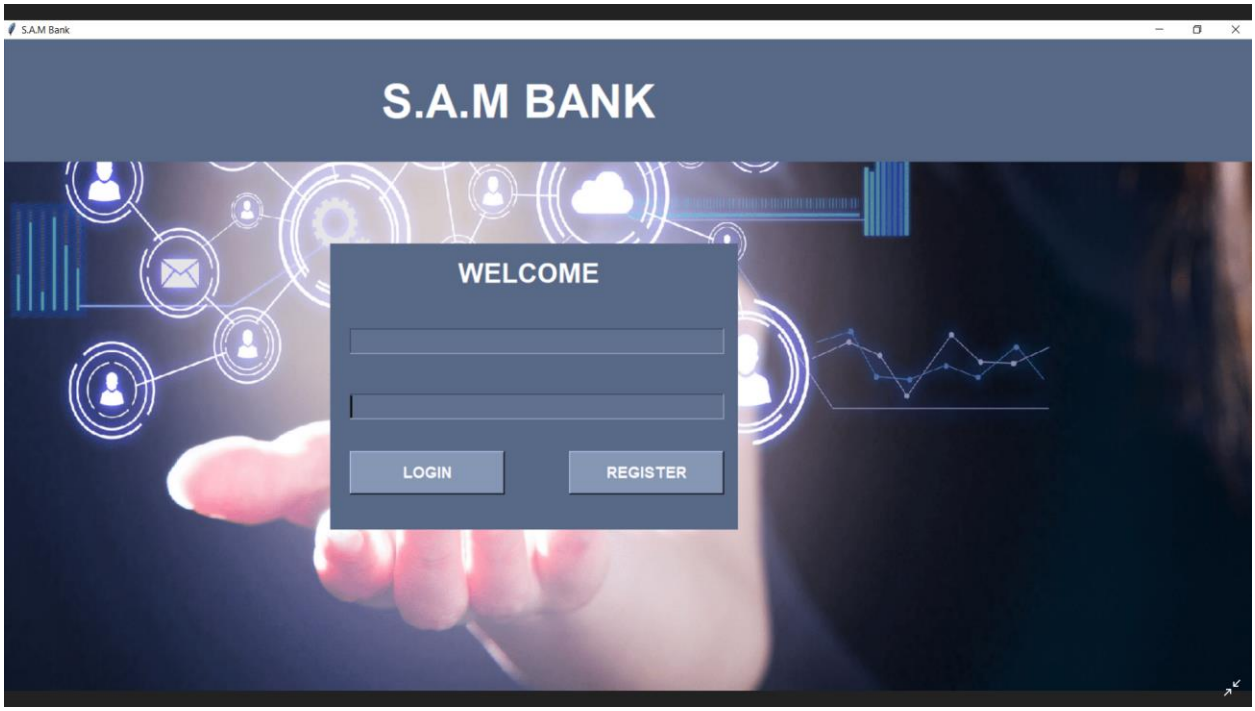
Test summary report:

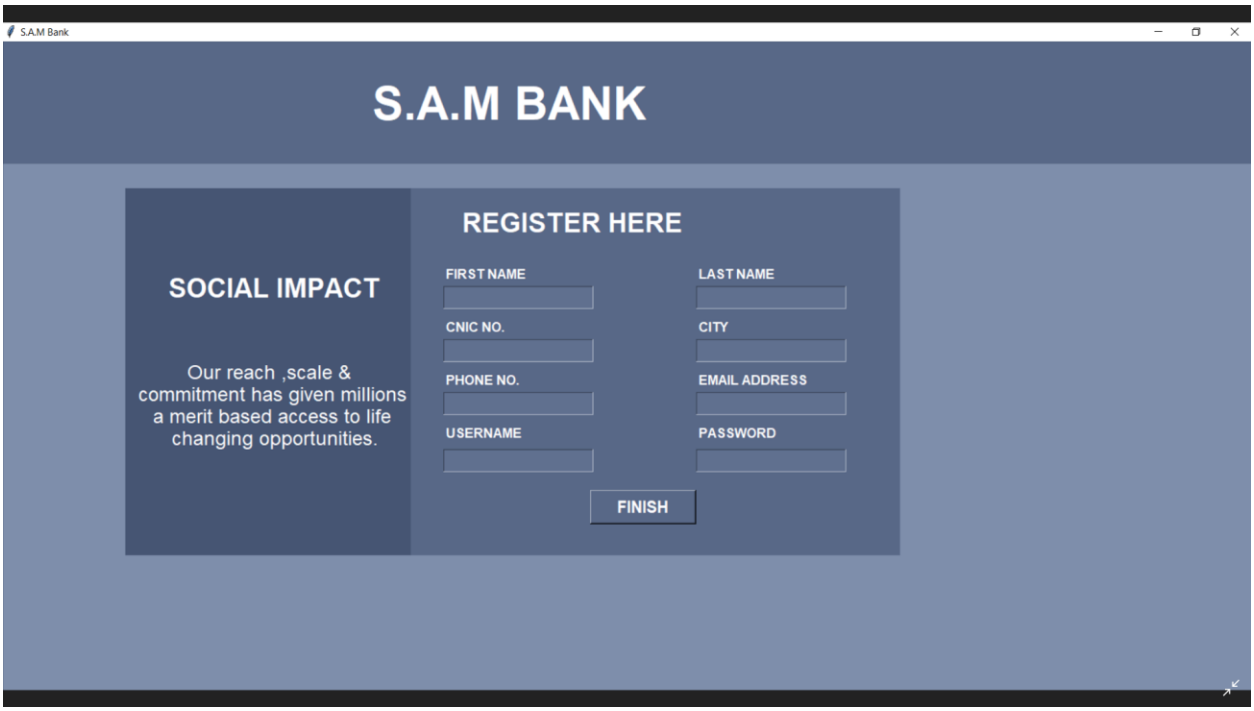
Project Name: Bank Management System
Module Name: Balance Check
Created By: Musadique Hussain
Date: 16/6/24
Test Scenario ID: BAL-001
Test Scenario Description: Verify user can check the balance

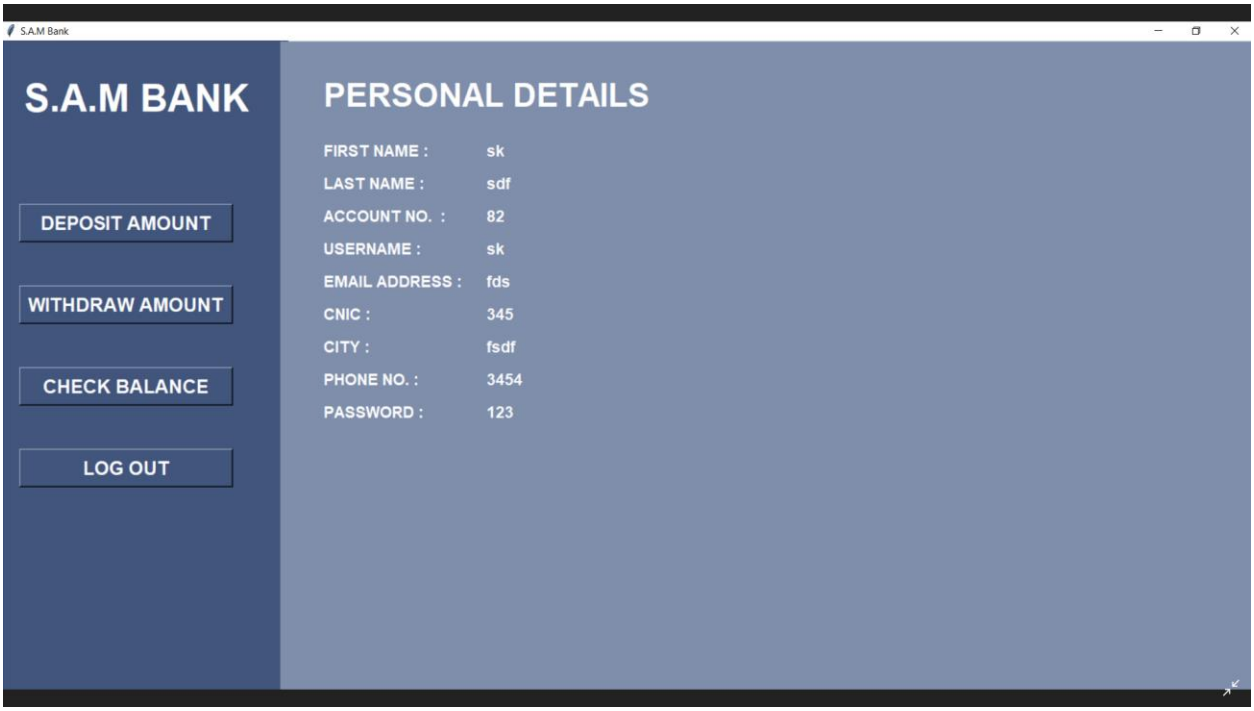
Test Case ID	Description	Steps	Test Data	Expected Output	Actual Output	Status
BAL - 01-01	Check the GUI	Open the application and navigate to registration form	N/A	All elements should be properly aligned	As expected	Pass
BAL - 01-02	Login with valid credentials	Enter valid username and password, submit	Valid credentials	User logged in and dashboard displayed	As expected	Pass
BAL - 01-03	Login with invalid credentials	Enter invalid username and password, submit	Invalid Credentials	Show error message	As expected	Pass
BAL - 01-04	Login with missing fields	Leave username or password empty, submit	N/A	Show error for missing fields	As expected	Pass
BAL - 01-05	Login with SQL injection	Enter SQL injection code in username or password, submit	SQL injection code	Show error and prevent login	As expected	Pass

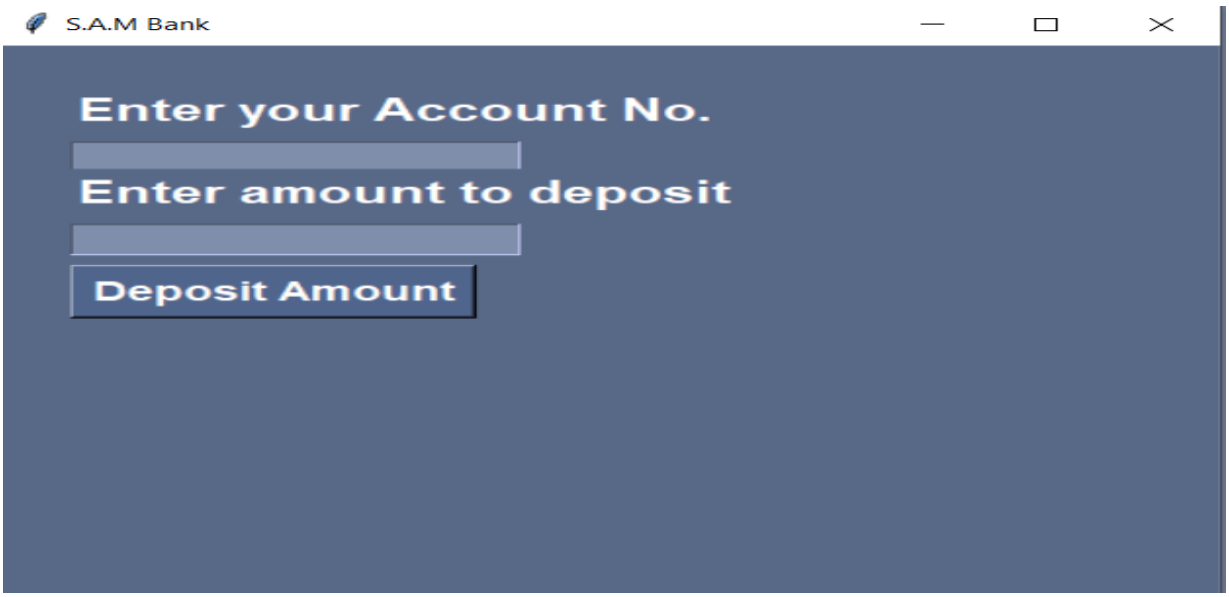
BAL - 01-06	Session management	Login, then check if session is maintained	Valid credentials	Session should be maintained correctly	As expected	Pass

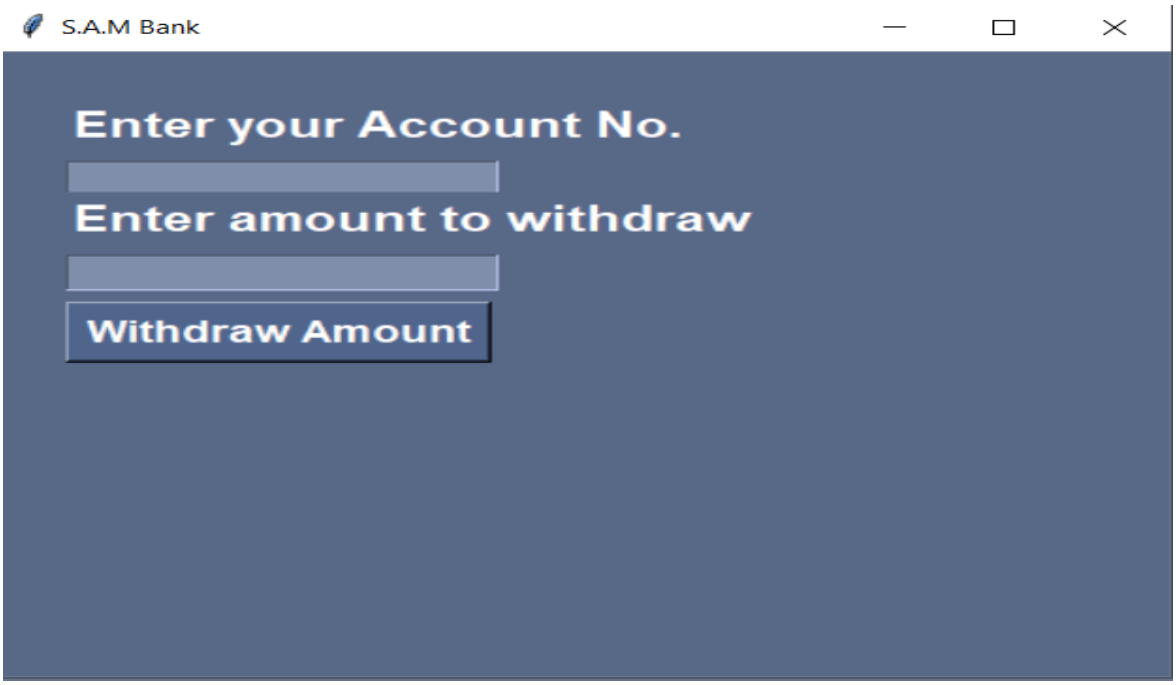
Application's Screenshots:




S.A.M Bank
S.A.M BANK
SOCIAL IMPACT
Our reach ,scale & commitment has given millions a merit based access to life changing opportunities.
REGISTER HERE
FIRST NAME
LAST NAME
CNIC NO.
CITY
PHONE NO.
EMAIL ADDRESS
USERNAME
PASSWORD
FINISH


S.A.M Bank
S.A.M BANK
DEPOSIT AMOUNT
WITHDRAW AMOUNT
CHECK BALANCE
LOG OUT
PERSONAL DETAILS
FIRST NAME : sk
LAST NAME : sdf
ACCOUNT NO. : 82
USERNAME : sk
EMAIL ADDRESS : fds
CNIC : 345
CITY : fsdf
PHONE NO. : 3454
PASSWORD : 123


S.A.M Bank
Enter your Account No.
Enter amount to deposit
Deposit Amount


S.A.M Bank
Enter your Account No.
Enter amount to withdraw
Withdraw Amount

Lab 9: Introduction to Katalon Studio + Installation

Katalon Studio is an automation testing solution, developed by Katalon, Inc. The software is built on top of the open-source automation frameworks Selenium, Appium with a specialized IDE interface for web, API, mobile and desktop application testing. Katalon Studio provides a dual interchangeable interface for creating test cases: a manual view for the less technical users and a script view gearing toward experienced testers to author automation tests with syntax highlight and intelligent code completion.

It follows the Page Object Model pattern. GUI elements on web, mobile, and desktop apps can be captured using the recording utility and stored into the Object Repository, which is accessible and reusable across different test cases. Test cases can be structured using test suites with environment variables. Test execution can be parameterized and parallelized using profiles. Remote execution in Katalon Studio can be triggered by CI systems via Docker container or command line interface (CLI). The test automation framework provided within Katalon Studio was developed with the keyword-driven approach as the primary test authoring method with datadriven functionality for test execution. The user interface is a complete integrated development environment (IDE) implemented on Eclipse rich client platform (RCP).

Katalon‟s main advantage is that it‟s easy to deploy and has a wider set of integrations compared to Selenium, the market leader. Katalon has dual scripting interfaces for users with different programming skills. This means that testers with limited technical knowledge can use a simpler user interface that doesn‟t require writing code. The mode for more proficient users has access to scripting with syntax highlighting, code suggestion, and debugging. Katalon Studio has pre-defined artifact structure: a number of templates for test cases, test suites, test objects, and reports. Katalon Studio supports local and remote testing, as well as parallel and sequential executions. It runs on Groovy (Java) scripting language. Katalon Studio is a cross-browser tool that supports Web, mobile, and API testing. These solutions go with analytics and recording modules. **Katalon Studio Key Features**

API testing

Unlike Selenium and Ranorex, with Katalon Studio you can test APIs without additional integrations. This inbuilt module allows testers to conduct end-to-end API testing, automate scripting, and maintain their tests. Main features of this module are code auto-completion, code inspection, code snippeting, references, and debugger. The software supports all types of REST, SOAP/1.1 and SOAP/1.2 requests and multiple data sources (XLS, XML, databases with dynamic mapping to maximize test coverage). The API testing mode allows users to import tests from such API testing and editing tools as Swagger, Postman, and WSDL. Katalon API testing also has an inbuilt response viewer with auto-formatting and search for access to artifacts.

Web testing

Katalon offers a complete web testing solution with inbuilt Continuous Delivery/Continuous Integration and DevOps integrations. It supports testing of Web, Android, and iOS applications. The tool has code-assist utilities like built-in object spy, code refactoring, and in-context references. Native plugins integrate with Jenkins, Git, JIRA, and qTest. Katalon‟s web-testing execution mechanism uses multiple configurations and datasets. It also allows for customizing execution workflow and running it automatically. The module features importing external libraries to improve the automation functions.

Mobile testing

This module allows users to test mobile Web, iOS, and Android apps. With native integration by means of Appium, a test automation tool for mobile applications, the module supports testing for the latest mobile platforms and devices without additional installations. Katalon speeds up mobile testing by detecting and storing the objects.

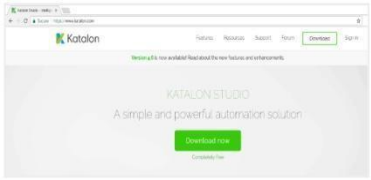
Supported technologies

- Modern web technologies: HTML HTML5, JavaScript, Ajax, Angular
- Windows desktop apps platforms:[21] Universal Windows Platform (UWP), Windows Forms (WinForms), Windows Presentation Foundation (WPF), and Classic Windows (Win32)
- Cross-browser testing: Firefox, Chrome, Microsoft Edge, Internet Explorer (9,10,11), Safari, headless browsers
- Mobile apps: Android and iOS (Native apps and mobile web apps)
- Web services: RESTful and SOAP

System Requirements:

- ✓ Operating System: Windows 7, Windows 8, Windows 10, macOS 10.11+
- ✓ CPU: 1 GHz or faster 32-bit (x86) or 64-bit (x64) processor
- ✓ Memory: 1 GB RAM (32-bit) or 2 GB RAM (64-bit)
  Hard Drive: at least 1 GB available hard disk space

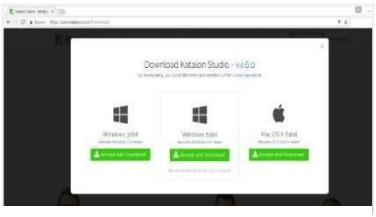
How to download and Install:



1. Go to Katalon Studio

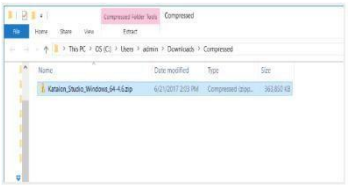
2. Click on "Download"

3. Give your full name, valid email, password, read and agree to the Katalon Privacy Policy and click on "Download

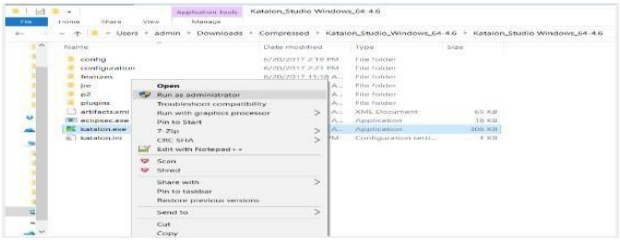


4. Choose a required download version and click on "Accept and Download"

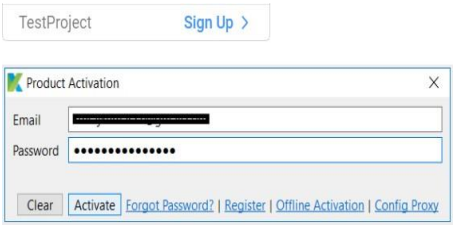
5. Once the download is complete – Go to the folder on your computer where the Katalon Studio is downloaded

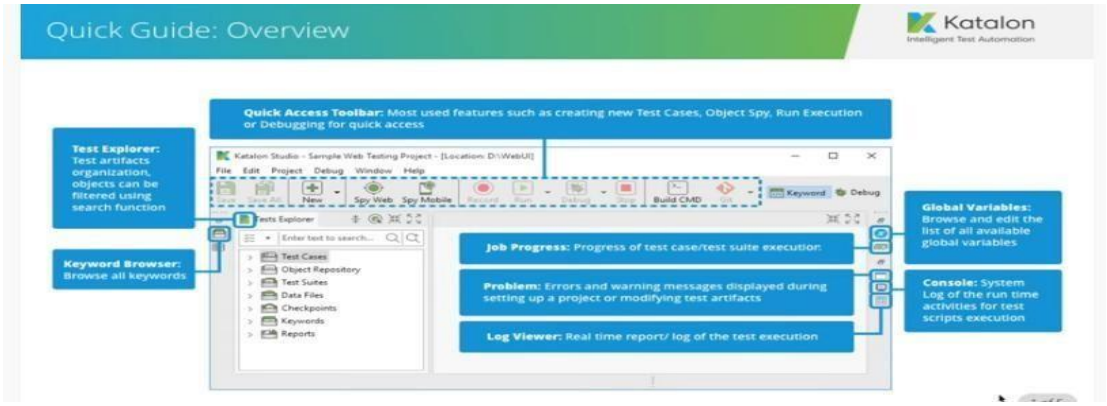


6. Run "Katalon.exe" to install Katalon Studio



7. Enter your account credentials that you used when signing up
8. Katalon launches and you are ready to automate.




Quick Guide: Overview
Katalon
Intelligent Test Automation
Quick Access Toolbar: Most used features such as creating new Test Cases, Object Spy, Run Execution or Debugging for quick access
Test Explorer: Test artifacts organization, objects can be filtered using search function
Keyword Browser: Browse all keywords
Global Variables: Browse and edit the list of all available global variables
Job Progress: Progress of test case/test suite execution
Problem: Errors and warning messages displayed during setting up a project or modifying test artifacts
Console: System Log of the run time activities for test scripts execution
Log Viewer: Real time report/ log of the test execution
Katalon Studio - Sample Web Testing Project - [Location: D:\WebUI]
File Edit Project Debug Window Help
New
Spy Web
Spy Mobile
Build CMD
Keyword
Debug
Tests Explorer
Enter text to search...
Test Cases
Object Repository
Test Suites
Data Files
Checkpoints
Keywords
Reports

Lab 10 : Explore Katalon GUI

Toolbar:The main **Toolbar** contains the most common actions which you usually perform (e.g. creating test resources or executing automation tests).

- ✓ Save the current opened test artifact
- ✓ Save all opened test artifacts.
- ✓ Create new test artifacts. You can select following options by selecting from dropdown list:

Folder
Test Case
Test Suite
Web Service Request

- •
- •
- •
- •
- •

- •
- • Test Data
- • Checkpoint
- Test Suite Collection
- Test Object
- • Package
- • Keyword

- ✓ Open **Object Spy** dialog for capturing elements on websites.
- ✓ Open **Mobile Object Spy** dialog for capturing elements on mobile applications.
- ✓ Open **Record** dialog for recording WebUI test cases.
- ✓ Open **Mobile Recorder** dialog for recording Mobile test cases.
- ✓ Run the current open test case. You can select these options by selecting from dropdown list:

- • Chrome
- • Firefox
- • IE
- • Safari
- • Edge
- • Remote
- • Headless
- • Android
- • iOS (on macOS)
- • Custom

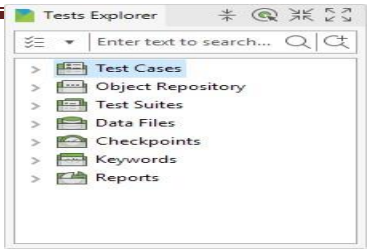
- ✓ Debug the current open test case. You can select these options by selecting from dropdownlist:

- • Chrome
- • Remote

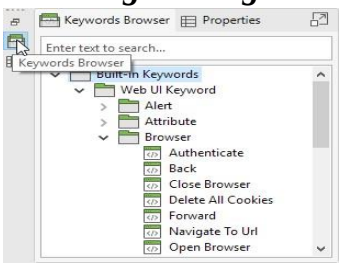
- Firefox
- Headless
- IE
- Android
- Safari
- iOS (on macOS)
- Edge
- Custom

- ✓ Stop the current execution
- ✓ Open **Command Builder** for generating commands for console execution
- ✓ Command for Git activities. You can select these options by selecting from dropdown list (after enabling Git):

Tests Explorer View: The Tests Explorer view allows you to browse the structure of your projects and access all test artifacts quickly. Using the context menu on the view, you can create new artifacts, organize the view's items or drag and drop them to certain editor views if needed.



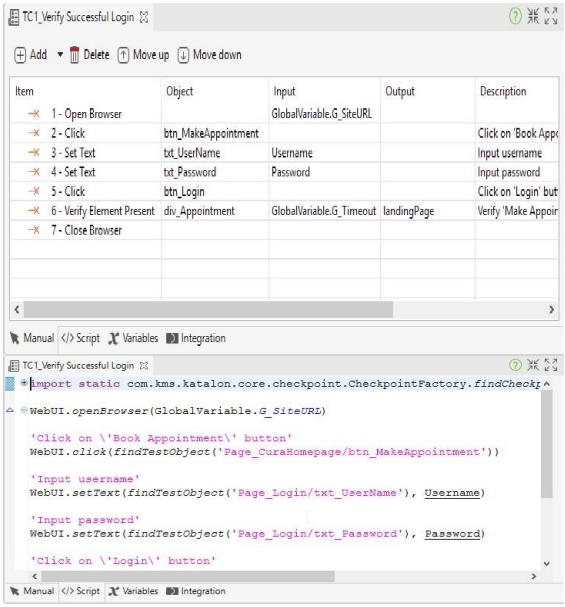
Group	Description
Test Cases	List all test cases within current project
Object Repository	List all test objects of the current project
Test Suites	List all test suites and test suite collections of the current project
Data Files	List all test data of the current project
Checkpoints	List all checkpoints of the current project
Keywords	List all custom keywords of the current project



Keywords Browser View: displays all available keywords supported by Katalon Studio. You can drag and drop the keywords here to the Test Case editor when scripting.

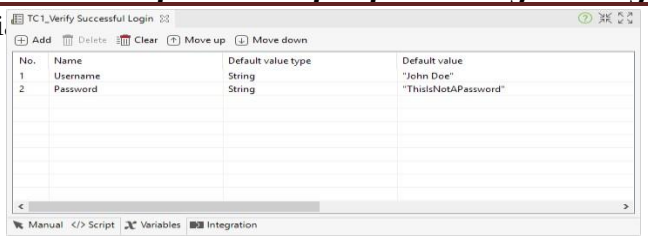
Editors: The editor is used to modify the detailed information of an object. Each test artifact has its own editor.

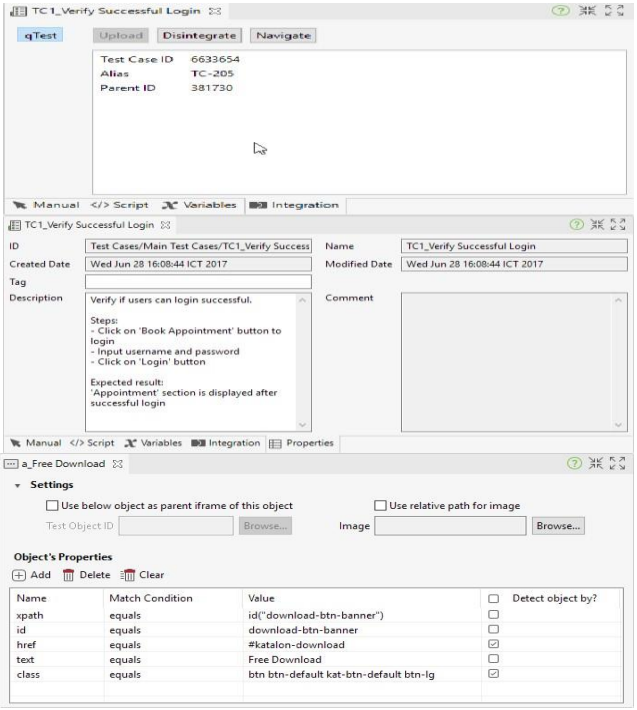
Test Case Editors: When you open a test case, its detailed information is shown in the editor that contains Manual tab, Script tab, Variables tab, Integration tab, Properties tab.

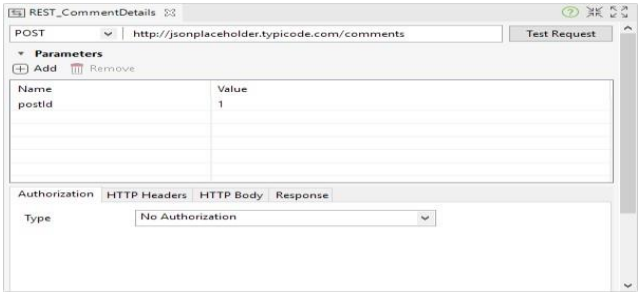


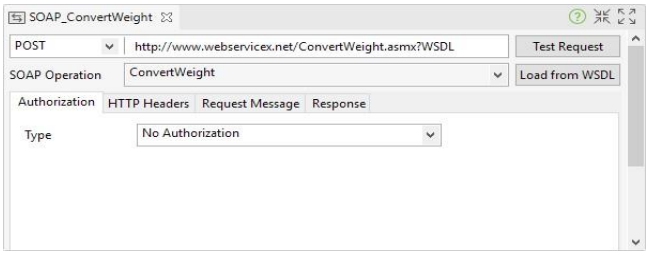
- ✓ **Manual Tab:** The Manual tab displays the Manual view where the basic keyword-driven- configuration allows novice users to create automation tests effortlessly.
- ✓ **Script Tab:** This tab displays the Script view where advanced users with programming background can modify test scripts easily using either Groovy or Java.

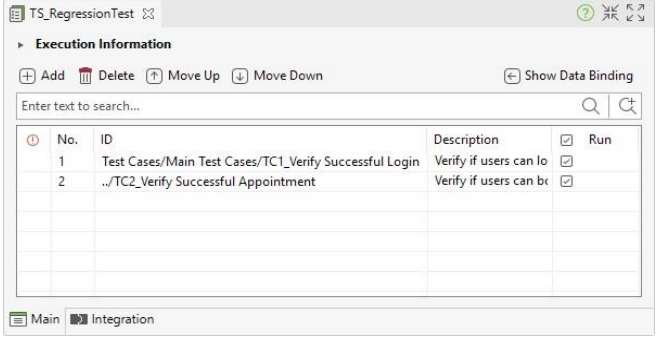
- ✓ **Variable Tab:** The Variables tab shows all vari











- ✓ **Integration Tab:** Information regarding your test case integration with qTest is displayed in this tab.

- ✓ **Properties Tab:** General information about the test case is displayed in this tab.

Test Object Editor: When you open a test object, its detailed information including properties and object identification mechanism is displayed in the Test Object editor.

Web Service Editor:

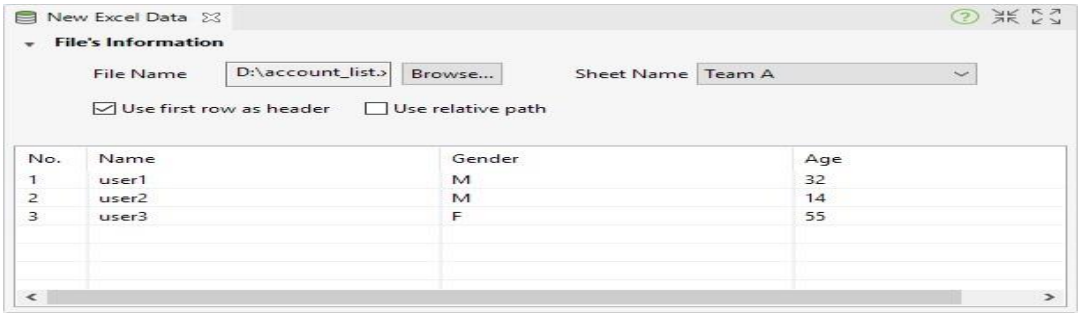
i. **RESTful Request Object Editor:**When you open a RESTful request object, its detailed information including the resource URL, request methods, parameters... is displayed in the editor.
ii. **SOAP Request Object Editor:** When you open a SOAP request object, its detailed information including the resource URL, request methods, parameters... is displayed in the editor
iii. ***Test Suite Editor:***When you open a test suite, its detailed information is displayed in the Test Suite editor that contains the Main tab **,**Integration tabs

Test Suite Collection Editor: a Test Suite Collection contains a set of test suites to allow users to execute test suites together. Refer to Execute a test suite collection for more details.

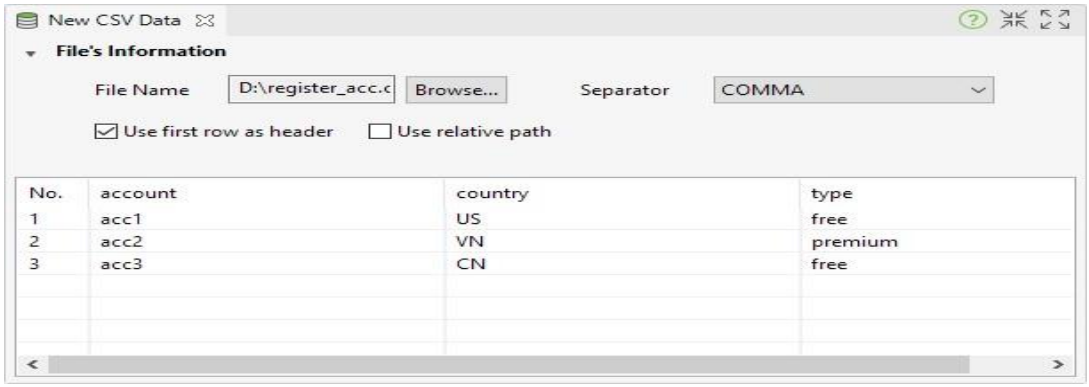
Data File Editor:When you open a data file, its detailed information including the data source and preview data set... is displayed in

the editor. Refer to Manage Test Data for more details.

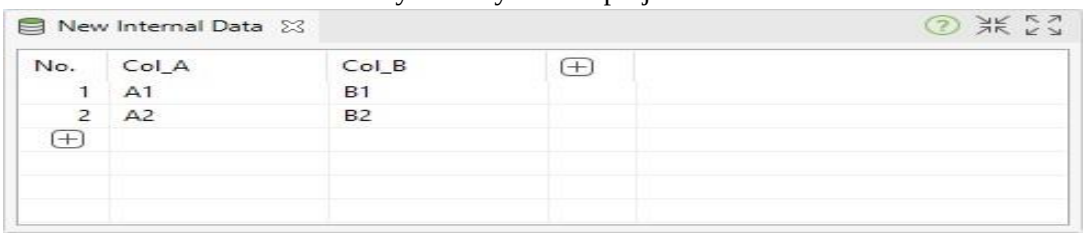
- If the test data is from an Excel file:



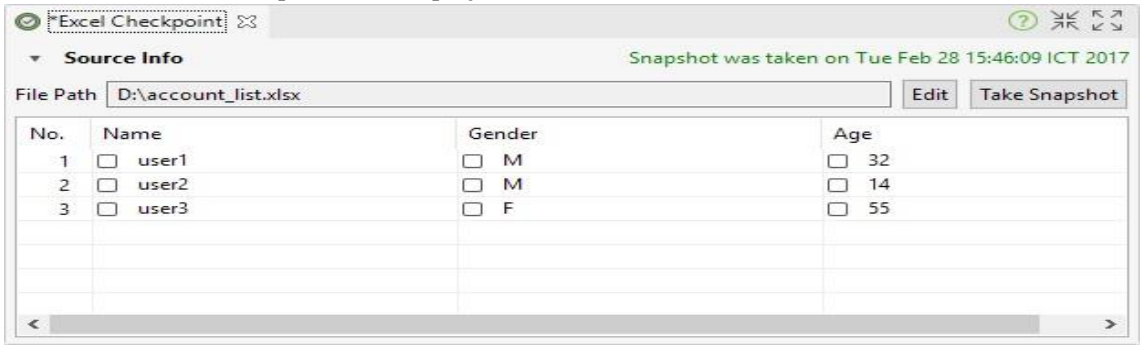
- If the test data is from a CSV file:



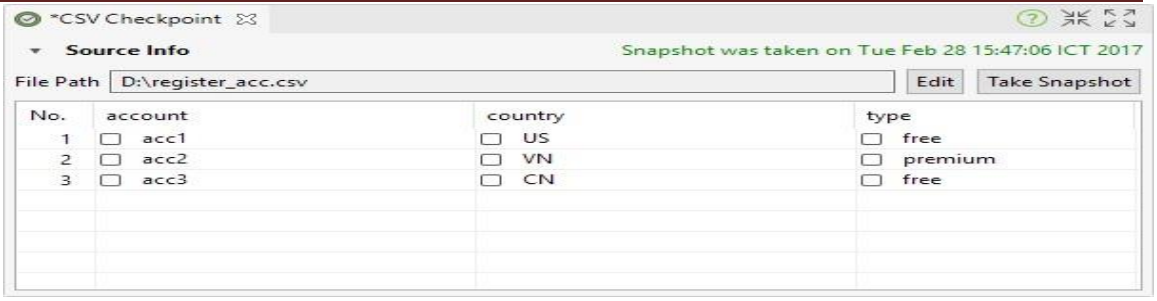
- If the test data is defined internally inside your test project:



Checkpoint Editor: When you open a checkpoint, its detailed information including the data source and its taken snapshot... is displayed in the editor. If the data source is from an Excel file:

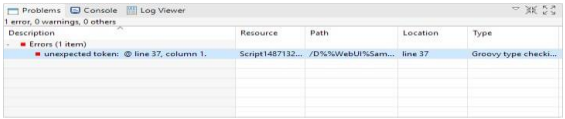


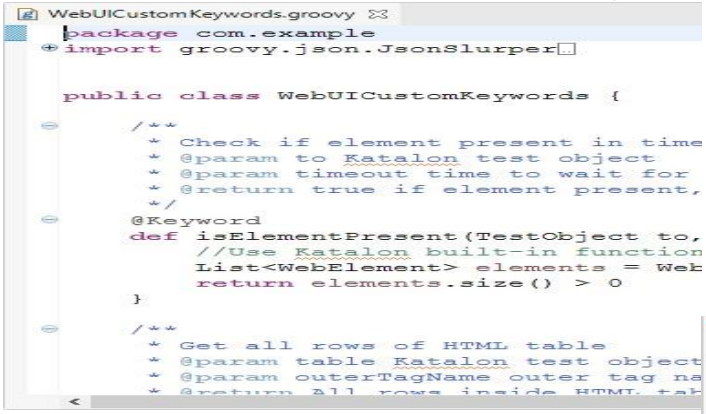
- If the data source is from a CSV file:



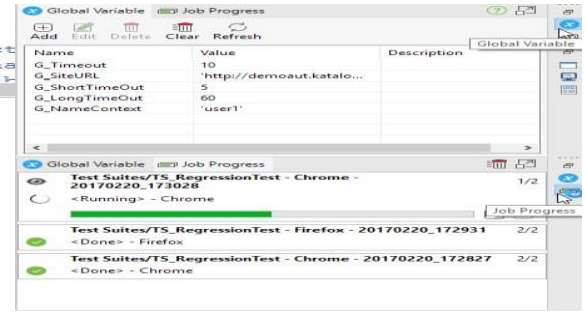
- If the data source is from a database:

Keyword Editor: When you open a custom keyword, its content is displayed in the scripting editor. This scripting editor is similar to the Script cases where you can define new keywords easily using Groovy or Java.





Global Variable View: The Global Variables view allows you to browse the list of available global variables defined in your project. Refer to Global Variables for more details.

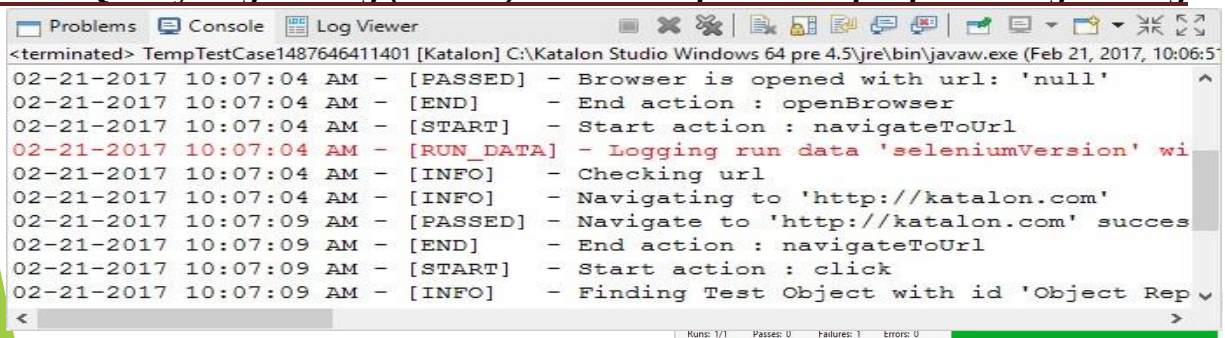


view of test custom

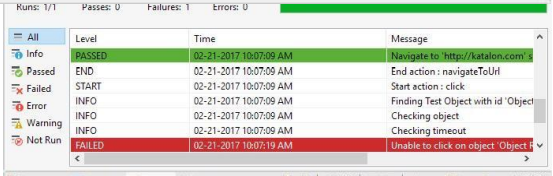
Job Progress View:The Job Progress view allows you to see the progress of executing test cases and test suites.

Problems View: The Problems view shows errors and warning messages raised when you are setting up a project or designing the test case, test suite, test object or test data.The messages displayed in the Problems view can be configured via the drop-down menu of this view.

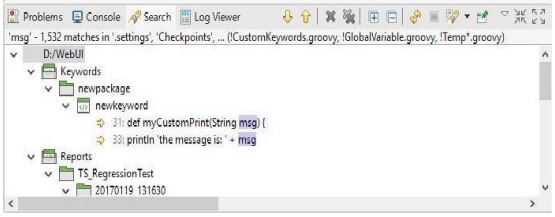
Console View: The Console view shows the system logs of all run-time activities performed while the automation test is being executed. The console output generated from test scripts is also displayed here.



Log Viewer View: The Log Viewer view shows the real - time report/log of the test execution.

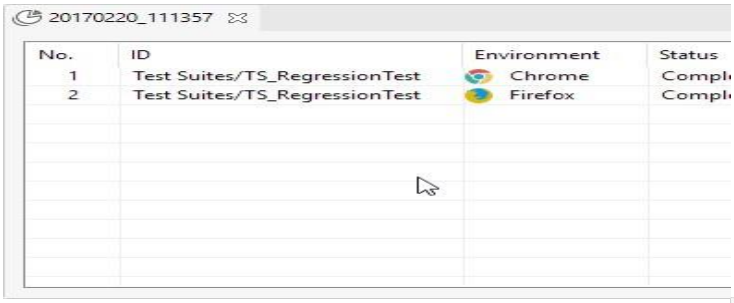


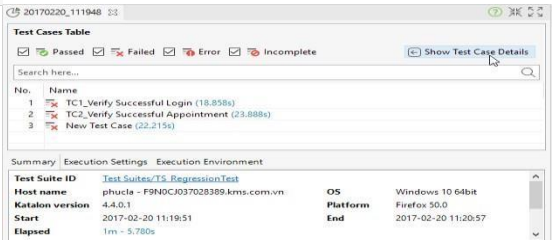
Search View: The Search view shows the search results from the search function. You can double-click on a search entry to navigate to the corresponding position in the editor.



Report View: The Report view allows you to view detailed information of a completed test execution for a certain test suite. Refer to Test Suite Report for more details.

Test Suite Collection Report View: The Test Suite Collection Report view allows you to view detailed information of a completed test execution for a certain test suite collection. Refer to Test Suite Collection Report for more details.





Who uses Katalon Studio?

- *Care Logistics*, a company that provides healthcare logistics software and services to hospitals and health centers, faced thechallenge of a quick verification of system stability. To verify the system's functionality, the testing team had to create over 8,000 manual test cases and execute all of them before deployment. With Katalon Studio, the team managed to convert and automate these tests.

  As a result, the team could verify more issues and had 30 percent of products automated. Also, they achieved a 50 percent reduction in regression testing lifecycle.
- *Tunaiku*, an Indonesian fintech lending venture, had to make sure that user data and transaction information was safe. This required a number of quality assurance tests in CI workflow, which involves Jenkins and Docker integration. Also, they needed to generate automation test scripts effectively. Tunaiku solved these quality assurance team problems

by creating automation test scripts and customized code infrastructure in Katalon. This helped them reduce scripting time by 60 percent.

- *Angler*, an Internet and technologies provider from India, used Katalon Studio for API testing. Their task was to test APIs and generate API scripts, as well as detect bugs in the development phase and prevent them from getting into APIs. As a result, Angler covered over 80 percent of functional testing with Katalon Studio.

Why Choosing Katalon Studio Over Selenium Open Source Framework:

- Katalon Studio hides all technical complexities behind the scene and provides friendly UI with the manual mode (user can drag-drop, select keywords and test objects to form test steps), but still keeps necessary weapons for more technically powerful users who are able to dig deeper into coding with the scripting mode that fully supports development conveniences like syntax highlighting, code suggestion and debugging
- With Selenium, users have to go through technical guidelines for their API and integrate them into with an IDE and a preferred programming language. By contrast, Katalon Studio is built as a unified bundle which includes almost all necessary things like Java, Android SDK, Web drivers to drive browsers, and required dependencies. All you have to do is to download and install it on your computer. Minimum installation is required
- While Selenium/Appium requires users to build common and reusable actions on their own, katalon Studio contains pre-defined sets of commonly used keywords or actions, users will find them enough to start implementing most test cases, but if they have more advanced or specific needs they still have a way to create custom keywords for their projects. Custom keywords are an extension of built-in keywords.

Why Choosing Katalon Studio Over QTP/UFT or Test Complete:

Katalon Studio scores fairly well as compared to commercial test automation tools such as UFT & Test Complete in multiple ways. It comes across as a viable, virtually free option to such tools available in the market:

Katalon Studio	**UFT/Test Complete**
No costs for licensing & maintenance	Cost-prohibitive solution: licensing & maintenance fees are quite high
Free upgrades	High fees for upgrades & add-on modules
Cross-platform solution	Only Windows based

Katalon Studio is a good choice for small and medium businesses. It‟s an evolving solution with many integrations that allow you to cover a variety of testing types with a single tool. It comes with all the necessary installations out of the box to run various types of tests, including API testing. It‟s free and easy to use. Katalon can be used by specialists with different QA engineering

roles and varied programming skills, making it an attractive solution for teams with testers of different levels. Katalon Studio is mostly used to automate functional testing to reduce time spent on scripting. That said, because it has downsides like minor bugs and support for only one scripting language, it‟s not the best enterprise solution.

EXERCISE:

1. **State the difference between Testcase & Test Suit**

Test case vs Test Suit

- A test case is a set of instructions that validate the functionality of a program or system. This will help to determine whether the code or system is functioning properly and satisfying the requirements.

- A test suite is a collection of test cases that have been grouped together for the purpose of test execution. It enables us to categorize and organize test cases in away that meets the requirements for planning and analysis.

2. **Why you preferred Katalon over any other Automation tool available in market ? state in your words also justify your answer with suitable example.**

Katalon Studio Over Automation Tool

▪ Katalon Studio is a cross-browser testing tool that works with Web, mobile, desktop, and API testing. These solutions include modules for analytics and recording. E.g. for a software consist of website as well as mobile app interface you can use a single automation testing tool for both web and app testing.

▪ It provides dual scripting interfaces for users with different programming expertise. As a result, testers with limited technical expertise can perform testing with a user friendly interface that doesn't require them to write code or have extensive vast technical knowledge.
E.g. Any member of a developing team can use it with no programming knowledge requirement.

▪ It is easy to setup and use. E.g. Tester from a team can use a number of pre-installed templates that allow repeating some testing patterns.

▪ It has many DevOps / ALM integrations, built-in object repository and other built-in features. It does not require licensing and maintenance fees.

3. **State the different views of Katalon GUI.**

Views of Katalon GUI

1. **Toolbar:** The main toolbar contains the most common actions like save, spy web, record web, run, debug, etc.

2. **Tests Explorer View:** The test explorer view allows to navigate to different test elements such as test case, test suits, checkpoints, and keyword etc. It also allows to create, organize and drag and drop testing artifacts.

3. **Keywords Browser View:** This view shows all the available keywords supported by katalon studio.

4. **Editors:** Editor is used to modify the detailed information of the test artifacts. Following are the editor views for each artifact:

Test Case Editor	For viewing, designing and do other settings of a test case
Test Object Editor	Displays detailed information and properties of test object
Web Service Editor	Displays detailed information including the resource URL, request methods and parameters of REST full and SOAP object requests
Test Suite Editor	For displaying and designing the test suit
Test Suite Collection Editor	For creating a collection of test suits which execute together
Data File Editor	Displays detailed information including the data source and preview data set of data file.
Checkpoint Editor	Displays detailed information including the data source and taken snapshot of data file.
Keyword Editor	Scripting editor for displaying the content of custom keyword

5. **Global Variables View:** This view enables us to browse through the collection of available global resources inside your project.

6. **Job Progress View:** This view enables us to see the progress of execution of test cases and test suits.

7. **Problems View:** This view enables us to see the warnings and error messages raised during project setup or designing the test case, test suite, test object or test data.

8. **Console View:** This view displays the system logs of all run-time activities performed while automation testing. Console output generated by test scripts is also displayed here.

9. **Log Viewer View:** This view displays the test execution report/log in real time.

10. **Search View:** This view displays the search results obtained using the search function. To navigate to a position in the editor, double-click on the corresponding search entry.

11. **Report View:** This view enables us to see the detailed information of a completed test execution for a certain test suite.

12. **Test Suite Collection Report view:** This view enables us to see the detailed information of a completed test execution for a certain test suite collection.

4. How many types of files you can attach to Katalon ? which feature of katalon allows you to attach file in katalon test suit.

Data files in Katalon Studio

We can attach four types of file to Katalon. These are: Excel file, CSV file, Database data, and internal data. The data binding feature of katalon allows us to attach file in katalon test suit.

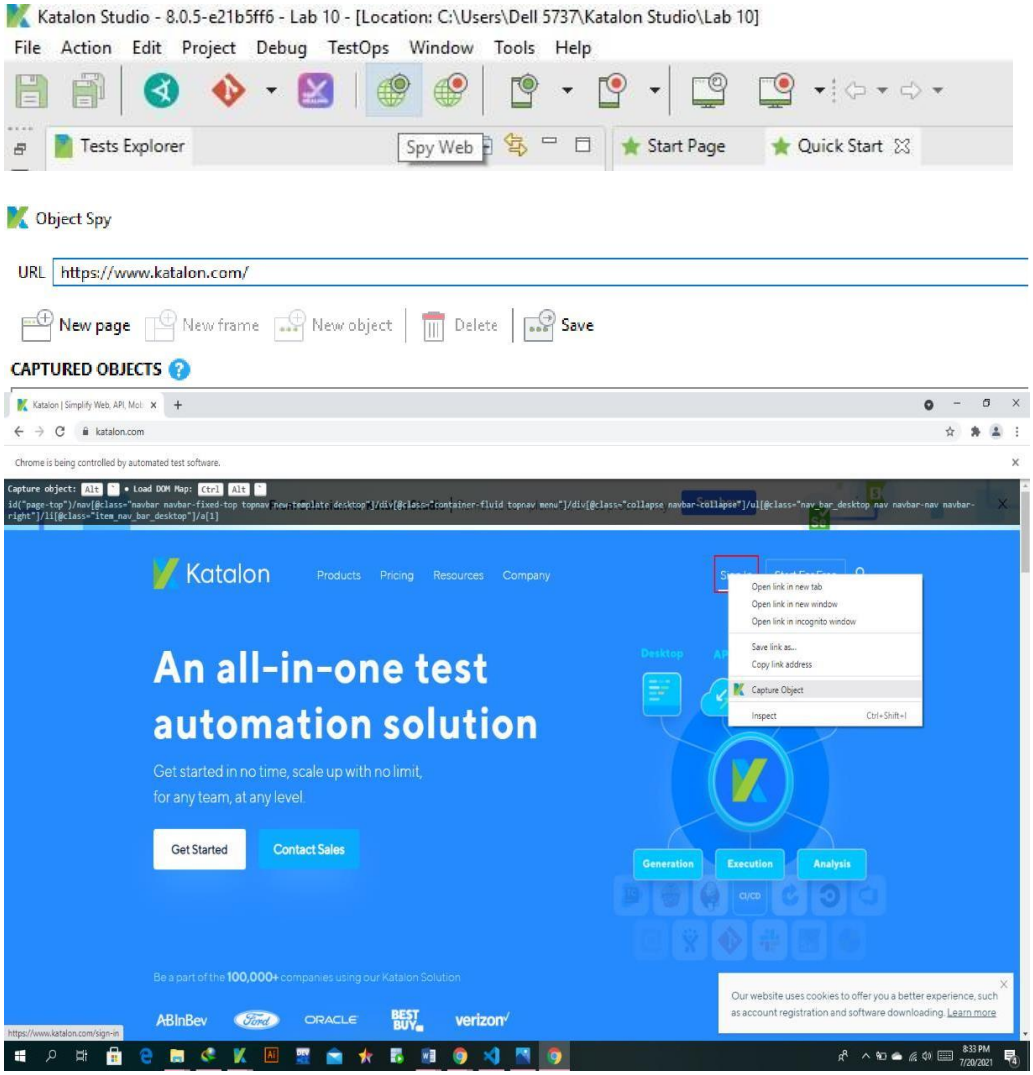
5. what is the purpose of Object Spy and Object Repository ? Explain with suitable example

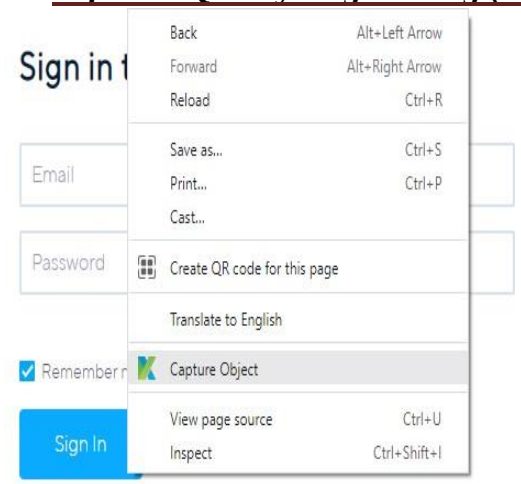
Object Spy and Objective Repository

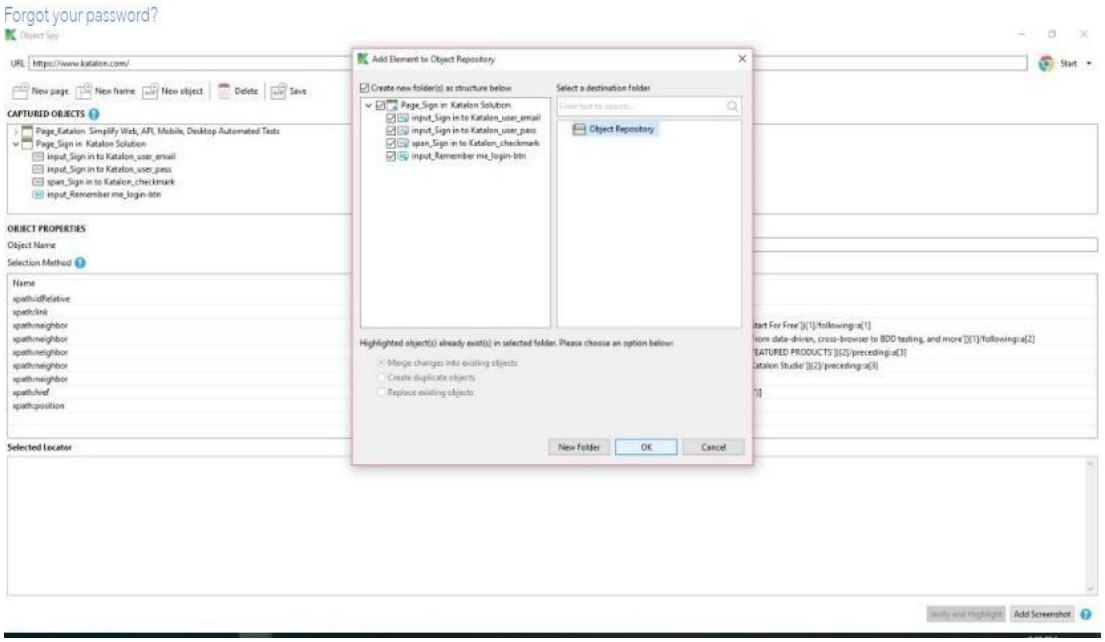
Object Spy enables us to capture the object from the software that is being tested to prepare the test cases. Whereas an object repository use to organize the repositories of object captured from the application.

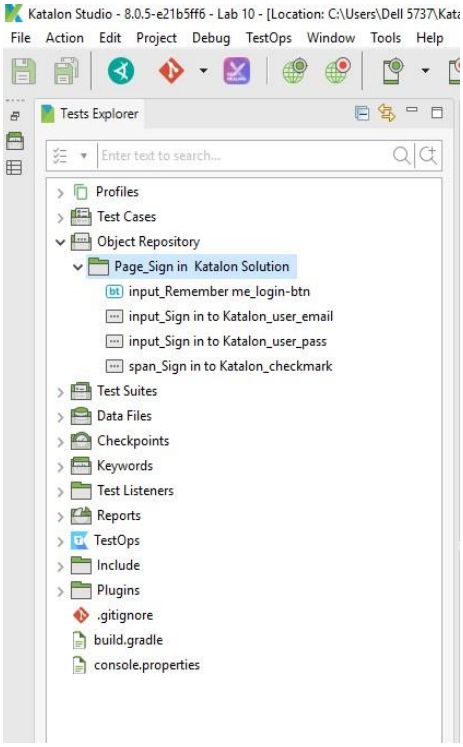
Example:

We capture the objects from **(https://www.katalon.com/)** for sign in test case using object spy and then save in object repository.


Katalon Studio - 8.0.5-e21b5ff6 - Lab 10 - [Location: C:\Users\Dell 5737\Katalon Studio\Lab 10]
File Action Edit Project Debug TestOps Window Tools Help
Tests Explorer
Spy Web
Start Page
Quick Start
Object Spy
URL https://www.katalon.com/
New page
New frame
New object
Delete
Save
CAPTURED OBJECTS
Chrome is being controlled by automated test software.
Katalon
Products
Pricing
Resources
Company
An all-in-one test automation solution
Get started in no time, scale up with no limit, for any team, at any level.
Get Started
Contact Sales
Open link in new tab
Open link in new window
Open link in incognito window
Save link as...
Copy link address
Capture Object
Inspect
Generation
Execution
Analysis
Be a part of the 100,000+ companies using our Katalon Solution
Our website uses cookies to offer you a better experience, such as account registration and software downloading. Learn more


Sign in t
Email
Password
Remember
Sign In
Back Alt+Left Arrow
Forward Alt+Right Arrow
Reload Ctrl+R
Save as... Ctrl+S
Print... Ctrl+P
Cast...
Create QR code for this page
Translate to English
Capture Object
View page source Ctrl+U
Inspect Ctrl+Shift+I


Forgot your password?
Add Element to Object Repository
Create new folder(s) as structure below
Select a destination folder
Object Repository
CAPTURED OBJECTS
OBJECT PROPERTIES
Selected Locator
New Folder
OK
Cancel


Katalon Studio - 8.0.5-e21b5ff6 - Lab 10 - [Location: C:\Users\Dell 5737\Kata
File Action Edit Project Debug TestOps Window Tools Help
Tests Explorer
Enter text to search...
Profiles
Test Cases
Object Repository
Page_Sign in Katalon Solution
input_Remember me_login-btn
input_Sign in to Katalon_user_email
input_Sign in to Katalon_user_pass
span_Sign in to Katalon_checkmark
Test Suites
Data Files
Checkpoints
Keywords
Test Listeners
Reports
TestOps
Include
Plugins
.gitignore
build.gradle
console.properties

Lab 11 A: Create Test Case : In Manual View

Katalon Studio supports Keywords-Driven testing where test cases consist of keywords that represent actions of users on the AUT (Applications Under Test). This allows users with less experience in programming to easily generate automation test. The below tutorial will give you step-by-step instruction in order to create an automation test case in manual mode.

Given a sample test case with the steps as below:

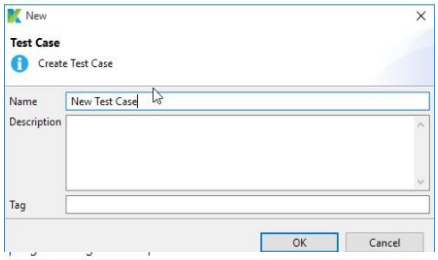
- *Open the browser*
- *Navigate to a website*
- *Click on certain control*
- *Validate if a control exists on the page*
- *Close the browser*

Follow these steps to automate the above test scenario in **Manual view**:

1. Select **File > New > Test Case** from the main menu. The **New Test Case** dialog will be displayed. Provide the name for the new test case, then click **OK** button.

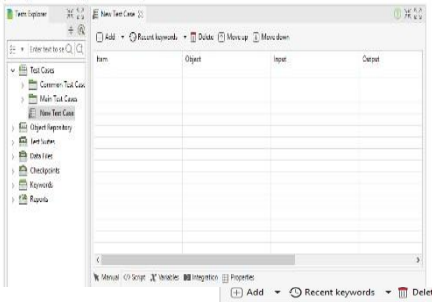
2. Once a new test case is created, it is opened in **Manual view**. This view allows users to create automation tests easily with little programming skills required.

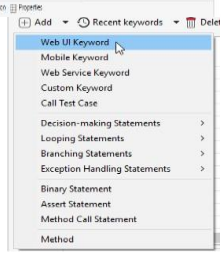
3. Select **Add > Web UI Keyword** from the command toolbar. Select the Open Browser keyword. This keyword will open a browser and navigate to the specified URL if provided. (selected keywords will have their description displayed along for reference)

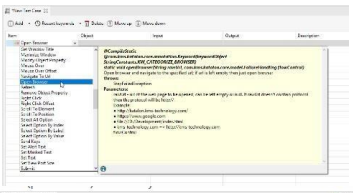


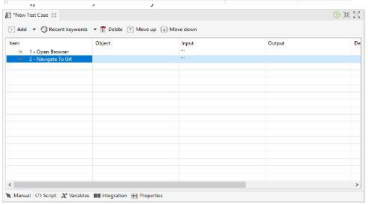
4. Add the Navigate To Url keyword. This keyword will navigate to a specified URL. Double click on the **Input** cell to provide additional data (parameters) for the keyword.

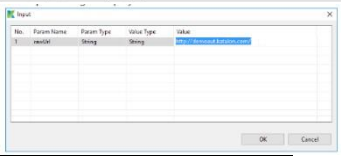
5. The Input dialog is displayed.









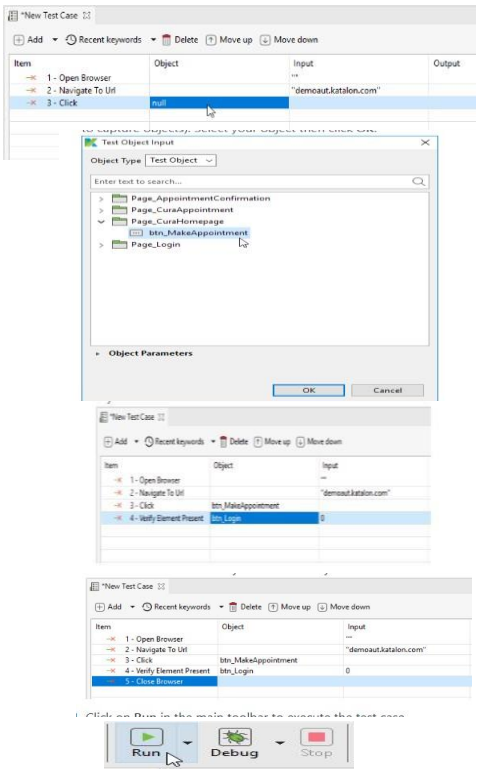


Where:

Field	**Description**
No.	The number of parameter for the selected keyword.
Param Name	The name of the parameter.

Param Type	The required data type for the parameter.
Value Type	The type of your input value (e.g.strings, variables, data sources...)

6. Add the **Click** keyword. This keyword represents the click action on a given object. Double click on the Object cell to provide the object for the keyword**.**



7. All captured objects in Object Repository are displayed in the Test Object Input dialog (Refer to **Spy Object** for details regarding how to capture objects). Select your object then click OK.

8. Add the **Verify Element Present** keyword. This keyword validates if a certain object is displayed on the executing browser. Similar to the previous step, you need to specify the object to be used with this keyword.

9. Add the **Close Browser** keyword and save your test case.

10. Click on Run in the main toolbar to execute the test case.

1. Write Testcases for login/logout features of "Orangehrm " website . Use username= Admin, Password=admin123 ; /*attach Test report and screenshot Here*/

a. Login:

Item	Object	Input
1 - Open Browser		""
2 - Navigate To Url		"https://opensource-demo.orangehrmlive.com/"
3 - Set Text	input_LOGIN Panel_txtUsername	""
4 - Set Text	input_Username_txtPassword	""
5 - Click	input_Password_Submit	
6 - Set Text	input_LOGIN Panel_txtUsername	"adm"
7 - Set Text	input_Username_txtPassword	"admin123"
8 - Click	input_Password_Submit	
9 - Set Text	input_LOGIN Panel_txtUsername	"Admin"
10 - Set Text	input_Username_txtPassword	"admin123"
11 - Click	input_Password_Submit	

b. Logout:

Item	Object	Input
1 - Click	a_Welcome Manoj	
2 - Click	a_Logout	

TEST REPORT:

Test Suite1

Execution Environment

Host name	NAJUM - NAJUM
OS	Windows 8.1 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

Summary

ID	Test Suites/Test Suite1		
Description			
Total	2		
Passed	2	Failed	0
Error	0	Incomplete	0
Start	2021-07-27 21:28:40	End	2021-07-27 21:29:17
Elapsed	37.617s		

#	ID	Description	Status
1	Test Cases/login		PASSED
2	Test Cases/logout		PASSED

2. Write Testcases (Using Manual tab) for My Info feature of "Orangehrm " website .Edit Nick name and save; Use at least One new keyword /*attach Test report and screenshot Here*/

a. Login:

Item	Object	Input
1 - Open Browser		""
2 - Navigate To Url		"https://opensource-demo.orangehr
3 - Set Text	input_LOGIN Panel_txtUsername	"Admin"
4 - Set Text	input_Username_txtPassword	"admin123"
5 - Click	input_Password_Submit	

b. Nickname:

Item	Object	Input
1 - Click	b_My Info	
2 - Click	input__btnSave	
3 - Set Text	input_Nick Name_personaltxtEmpNickName	""
4 - Click	input__btnSave	
5 - Click	input__btnSave	
6 - Set Text	input_Nick Name_personaltxtEmpNickName	"nickname"
7 - Click	input__btnSave	

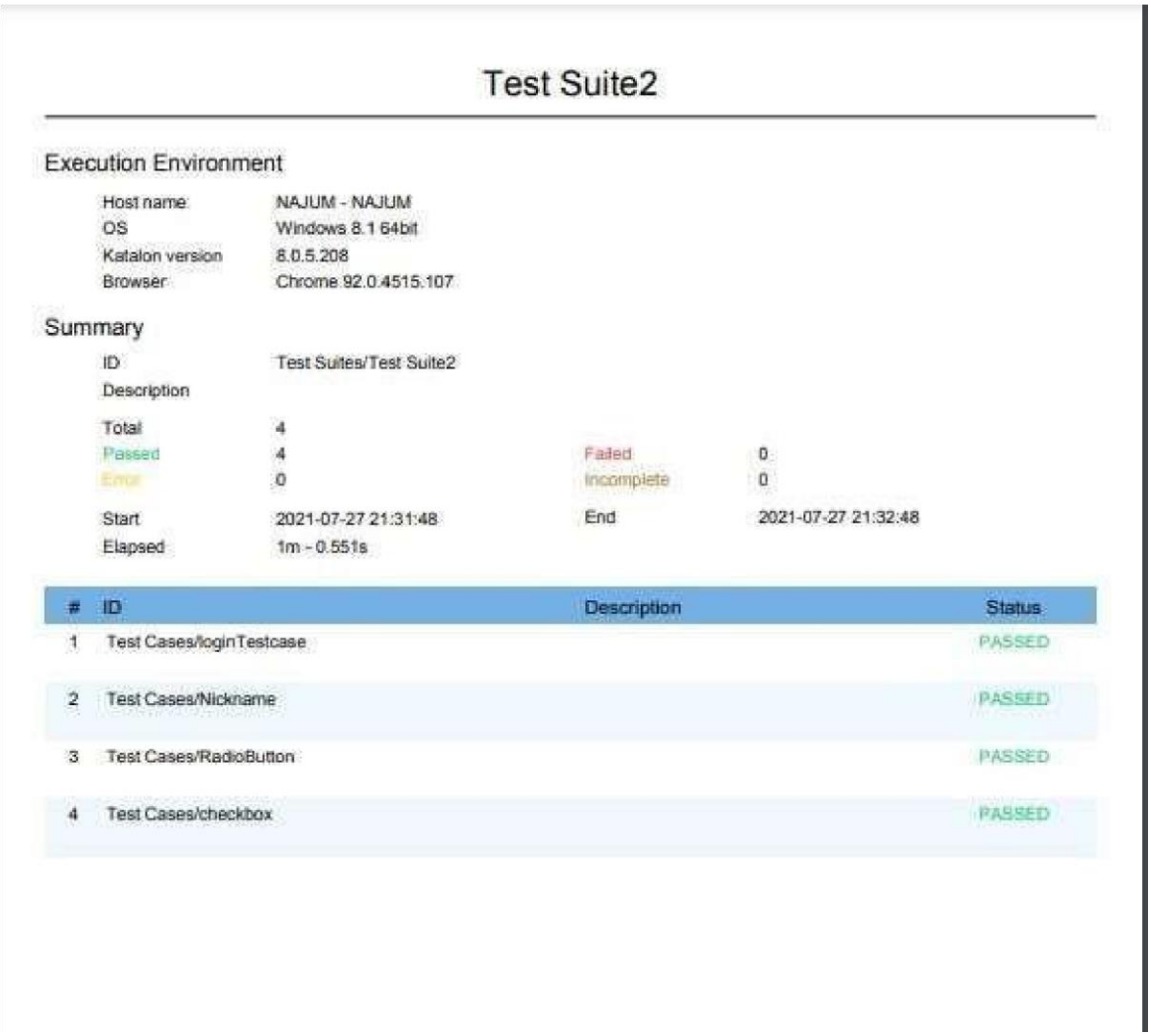
c. Radio Button:

Item	Object	Input
1 - Click	input__btnSave	
2 - Click	input_Gender_personaloptGender	
3 - Click	input__btnSave	
4 - Click	input__btnSave	
5 - Click	input_Male_personaloptGender	
6 - Click	input__btnSave	

d. Checkbox:

Item	Object	Input
1 - Click	input__btnSave	
2 - Check	input_Smoker_personalchkSmokeFlag	
3 - Click	input__btnSave	

TEST REPORT:



Test Suite2

Execution Environment

Host name	NAJUM - NAJUM
OS	Windows 8.1 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

Summary

ID	Test Suites/Test Suite2		
Description			
Total	4		
Passed	4	Failed	0
Error	0	Incomplete	0
Start	2021-07-27 21:31:48	End	2021-07-27 21:32:48
Elapsed	1m - 0.551s		

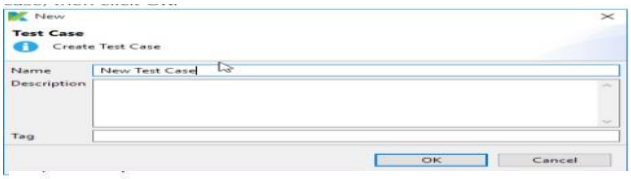
#	ID	Description	Status
1	Test Cases/loginTestcase		PASSED
2	Test Cases/Nickname		PASSED
3	Test Cases/RadioButton		PASSED
4	Test Cases/checkbox		PASSED

Lab 11 B: Create Test Case: Script View

Katalon Studio allows expert users to programmatically write automation test in the Script view of test cases. Users with Groovy/Java background can easily modify the test script as needed.

Given a sample test case with the steps as below:

- *Open the browser*
- *Navigate to a website*
- *Click on certain control*
- *Validate if a control exists on the page*
- *Close the browser*



Follow these steps to automate the above test scenario in **Script view**:

1. Select **File > New > Test Case** from the main menu. The **New Test Case** dialog will be displayed. Provide the name for the new test case, then click **OK**.

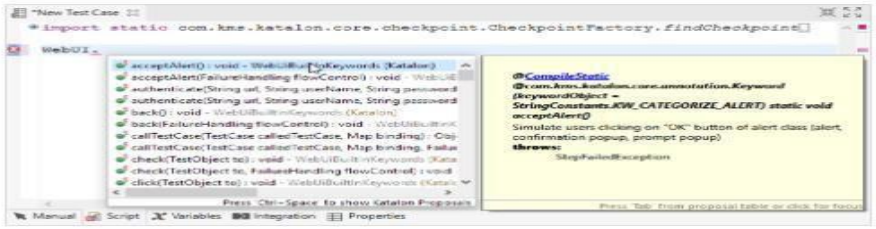
2. Once a new test case is created, you can switch to the Script view using the corresponding tab at the footer of the test case editor. Test steps specified in the Manual view are translated into a Groovy script in Script view.

The import statement in a test script allows referencing to classes to be used. Expand the **import** section to see all default imported classes by Katalon Studio. The name after 'as' in each import statement is an alias for the class. You can change the alias for each class. These classes are necessary for composing a test script. Katalon Studio is an automation tool that supports keyword-driven testing. All keywords are grouped into WebUI, Mobile and WebService packages accordingly. Press **Ctrl + Space** to view these packages and functions from the imported classes.

3. In this scenario, you will create a Web application test script to make use of the Web UI built-in keywords. To use a built- in

WebUI

The Content Assist function will be invoked after users enter the dot character. Content Assist provides users with a context-sensitive suggestion for code completion. Therefore, all the built-in keywords for WebUI testing will be displayed as below:

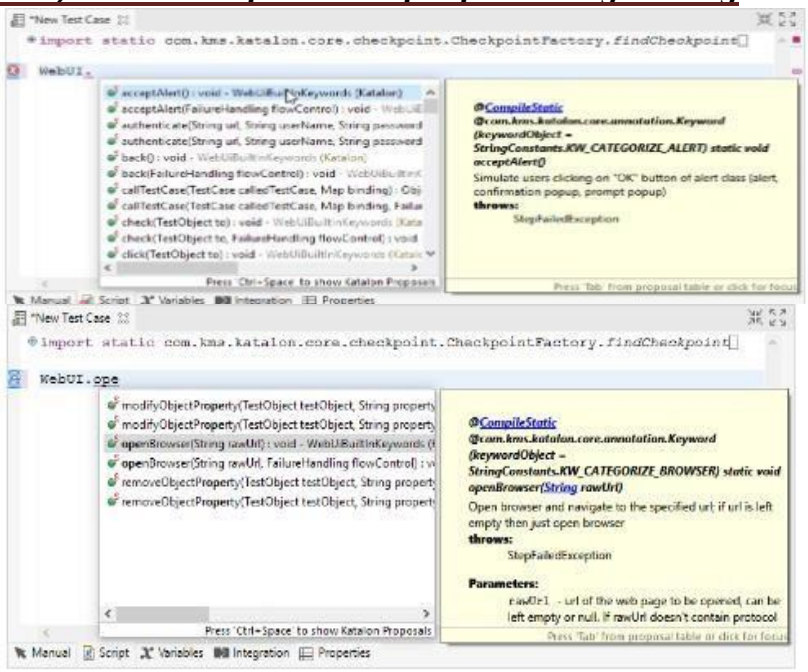


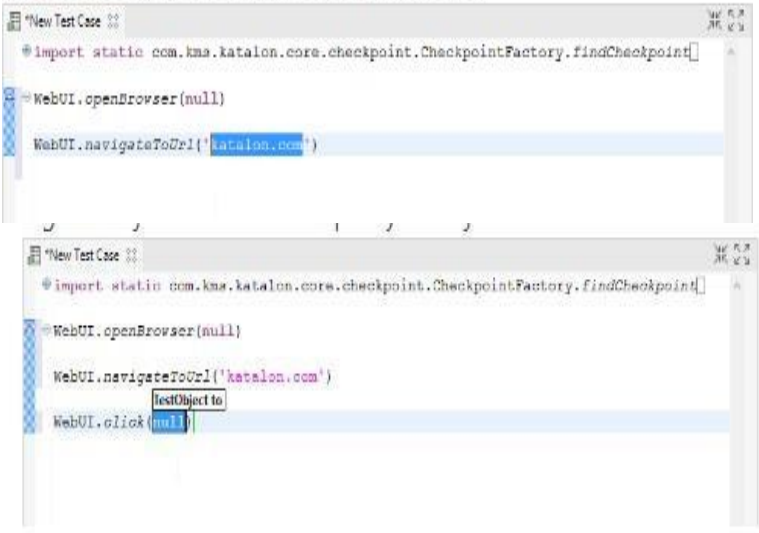
4. After entering the dot character (.), all built-in keywords and their description for Web UI testing will appear as shown in the figure:

5. Select the **Open Browser** keyword. This keyword opens a browser and navigates to the specified URL if it is provided. Selected keywords will have their description displayed along for reference.

6. Enter the **Navigate To Url** keyword. This keyword navigates to a specified URL. For now, enter the URL of Katalon Studio (**katalon.com**) as the value of the parameter.

7. Enter the **Click** keyword. This keyword represents the click action on a given object. You need to specify an object for this action.Use the following syntax to refer to an object in Object Repository (alternatively, you can drag and drop the object to test case editor to generate the syntax):





```
findTestObject('{Object ID}')
```

Where Object ID **is the ID of that object in Katalon Studio.**

9. Enter the **Verify Element Present** keyword. This keyword validates if a certain object is displayed on the executing browser. Similar to the previous step, you need to specify the object to be used with this keyword.

Add the **Close Browser** keyword and save your

10.

test case.

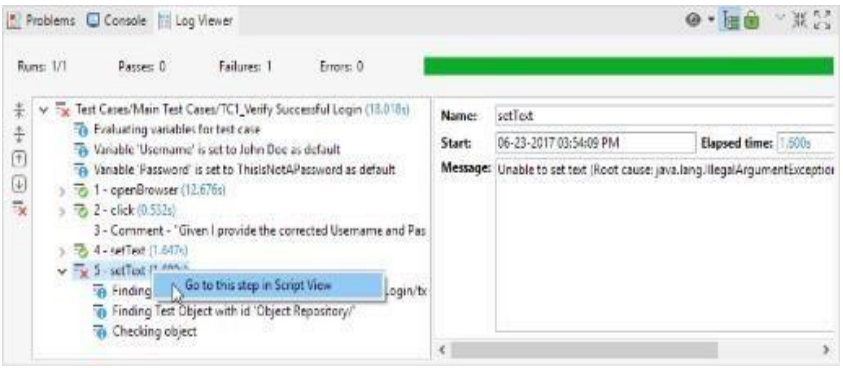
The following API docs may prove useful when working in Script view:

Class	Description
Builtin Keywords	List of common built-in keywords
WebUI Builtin Keywords	List of Web UI built-in keywords

Web Service Builtin Keywords	List of Web Service built-in keywords
Mobile Builtin Keywords	List of Mobile built-in keywords

- Click on Run in the main toolbar to execute the test case.

Katalon Studio should be able to execute all the steps of the sample test case.The test execution results are shown in **Log Viewer** as below:



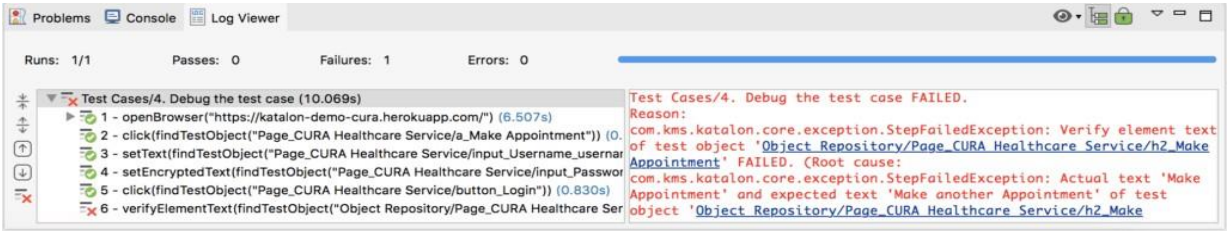
Verification in the test case: To get the result of the login process, you need to add the verification script to the test case. As the fundamental of scripting in Katalon Studio for the first test case has been explained, we can move on to another test case: "Make Appointment" header verification. You can leverage Katalon Studio‟s rich set of verification keywords to work on that requirement. Note that this kind of script can be done in both Manual and Script mode.

Item	Object	Input
1 - Open Browser		"https://katalon-demo-cura.herokuap
2 - Click	a_Make Appointment	
3 - Set Text	input_Username_username	"John Doe"
4 - Set Encrypted Text	input_Password_password	"g3/DOGG74jC3Flrr3yH+3D/yKbOqq
5 - Click	button_Login	
6 - Verify Element Text	h2_Make Appointment	"Make Appointment"
7 - Close Browser		

WebUI.verifyElementText(findTestObject('Object Repository/Page_CURA Healthcare Service/h2_Make Appointment'), 'Make Appointment', FailureHandling.STOP_ON_FAILURE)
You can then check the execution results in the Log Viewer

Debugging the test case: Try changing the verification text to "Make another Appointment." to make the test case fail. Here are a few options of how you can check the reasons for failure in Katalon Studio:

- Option 1: Investigate error logs



- Option 2: Debug mode

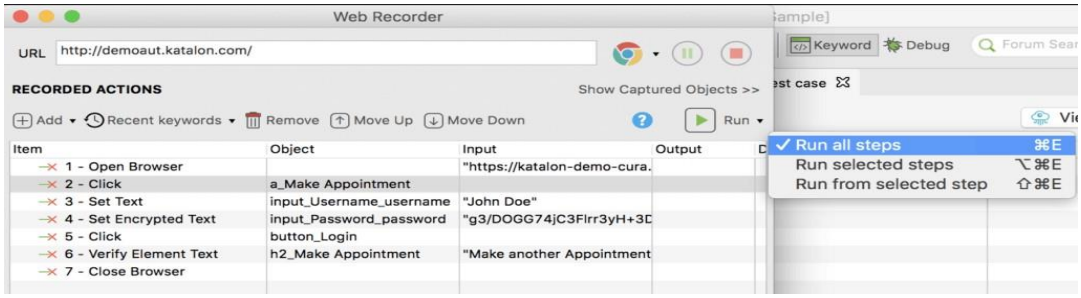
For complex cases, Katalon Studio offers a debug mechanism that works similarly to the code debug mechanism in the advanced developer IDE.

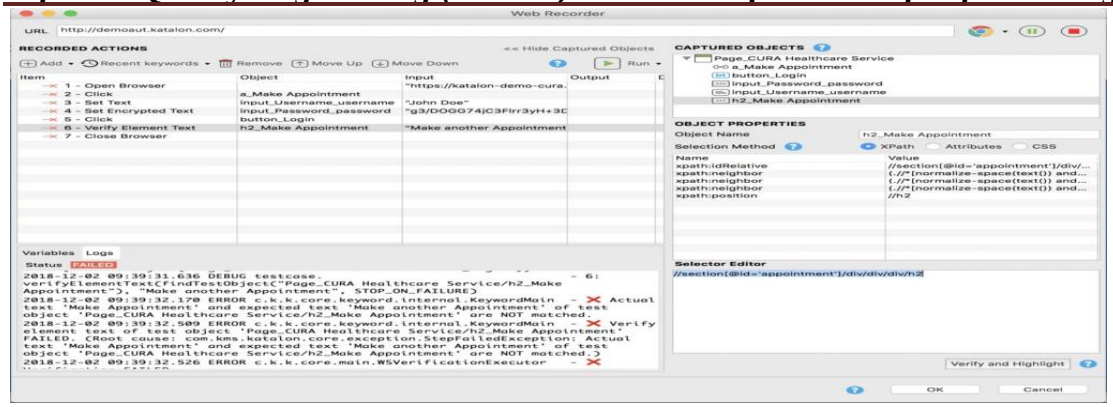
- Option 3: Manual debug

It is possible to use the „Recording" feature of Katalon Studio for debugging. The tool supports recording the failed test case, running the error test step for you to analyze and fix the issue directly in the recording mode.

When your test case failed, click "Yes, I do" in the window that shows up to continue recording the failed test cases.

Choose Run all steps to run the error test step. After that, you can investigate and fix the issue directly in the recording mode.





EXERCISE:

1. **Write TestCases for login/logout features of "Cura Healthcare "website through script. Also verify text" we care about your health "with true or false text values.;/* attach Test report and script here*/ <u>Script:</u>**

a. Log in

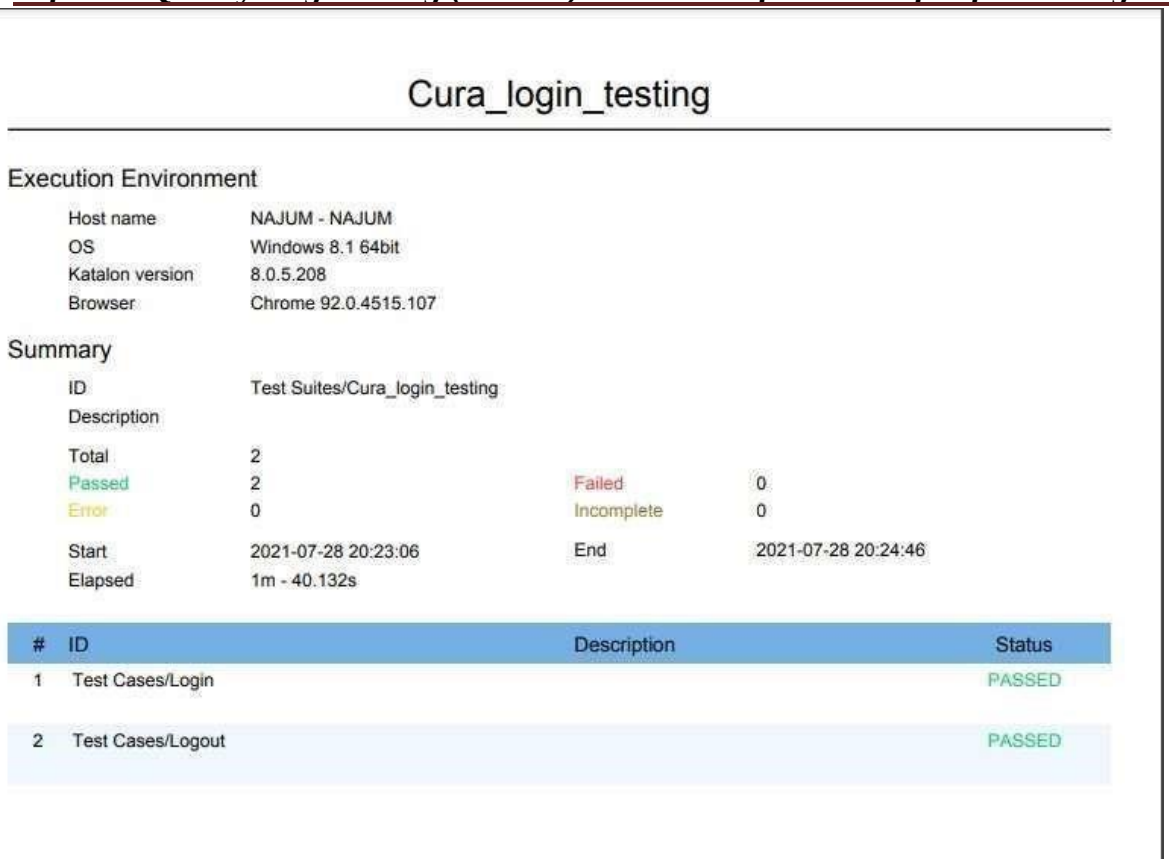
```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint

WebUI.openBrowser('')

WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com')

WebUI.click(findTestObject('Page_CURA Healthcare Service/i_CURA Healthcare_fa fa-bars'))

WebUI.click(findTestObject('Page_CURA Healthcare Service/a_Login'))

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Username_username'), '')

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Password_password'), '')

WebUI.click(findTestObject('Page_CURA Healthcare Service/button_Login'))

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Username_username'), '')

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Password_password'), 'ThisIsNotAPassword')

WebUI.click(findTestObject('Page_CURA Healthcare Service/button_Login'))

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Username_username'), 'John Doe')

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Password_password'), 'Thisispassword')

WebUI.click(findTestObject('Page_CURA Healthcare Service/button_Login'))

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Username_username'), 'John Doe')

WebUI.setText(findTestObject('Page_CURA Healthcare Service/input_Password_password'), 'ThisIsNotAPassword')

WebUI.click(findTestObject('Page_CURA Healthcare Service/button_Login'))
```

b. Logout

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint

WebUI.click(findTestObject('Page_CURA Healthcare Service/a_CURA Healthcare_menu-toggle'))

WebUI.click(findTestObject('Page_CURA Healthcare Service/a_Logout'))

WebUI.closeBrowser()
```

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint

WebUI.openBrowser('')

WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com')

WebUI.verifyElementText(findTestObject('Page_CURA Healthcare Service/h3_We Care About Your Health'), 'We Care About Your Health')

WebUI.verifyElementText(findTestObject('Page_CURA Healthcare Service/h3_We Care About Your Health'), 'We Care About Your healths')

WebUI.acceptAlert()

WebUI.closeBrowser()
```

Test Report



Cura_login_testing

Execution Environment

Host name	NAJUM - NAJUM
OS	Windows 8.1 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

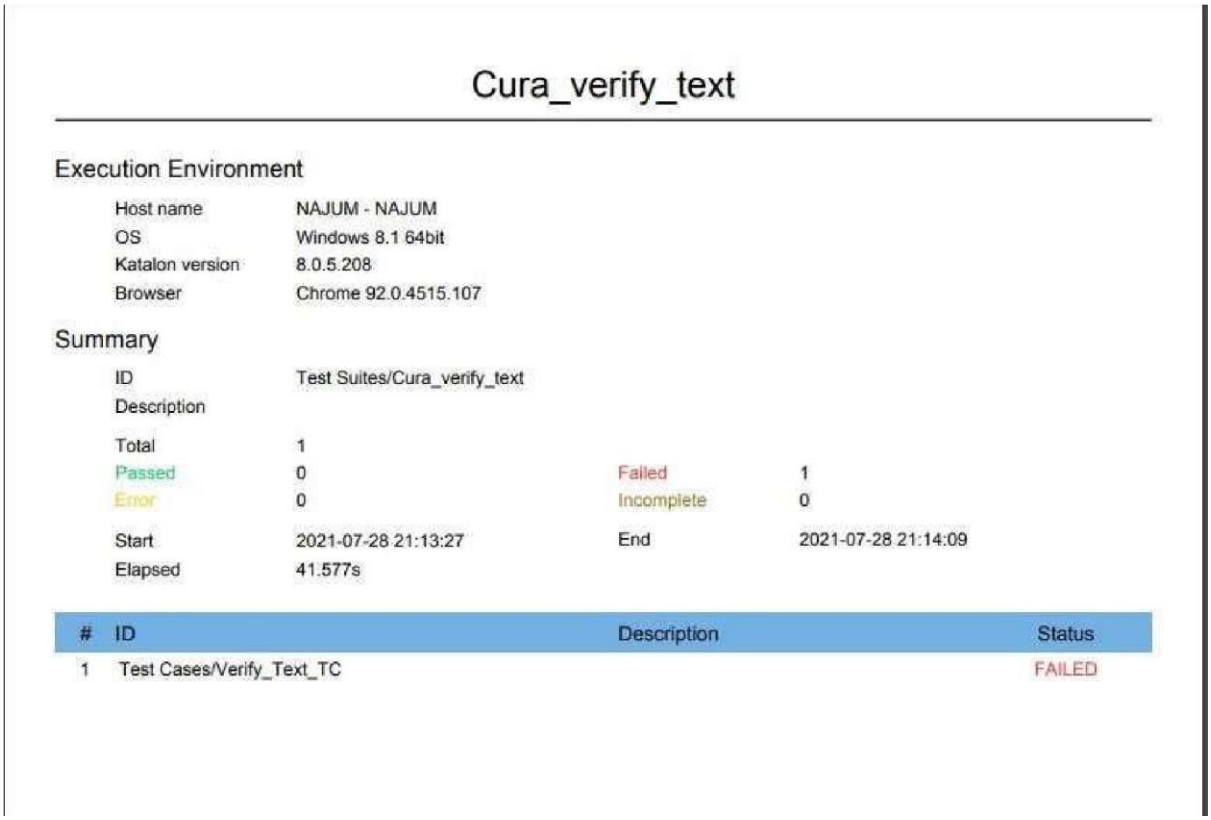
Summary

ID	Test Suites/Cura_login_testing		
Description			
Total	2		
Passed	2	Failed	0
Error	0	Incomplete	0
Start	2021-07-28 20:23:06	End	2021-07-28 20:24:46
Elapsed	1m - 40.132s		

#	ID	Description	Status
1	Test Cases/Login		PASSED
2	Test Cases/Logout		PASSED

Verify Text:

TEST REPORT:



Cura_verify_text

Execution Environment

Host name	NAJUM - NAJUM
OS	Windows 8.1 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

Summary

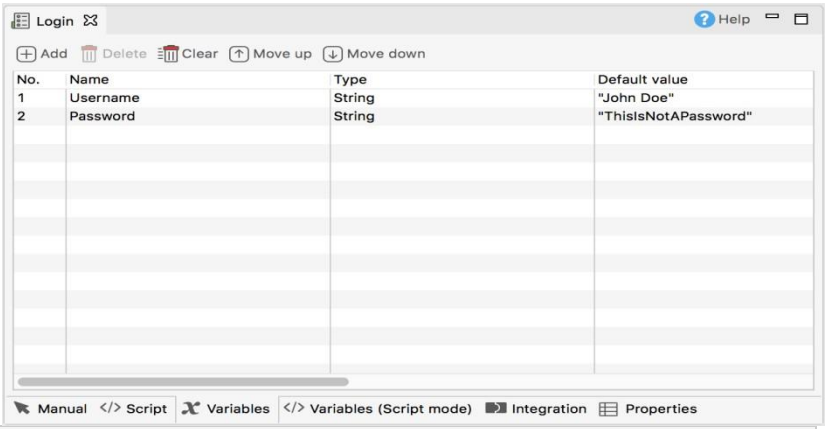
ID	Test Suites/Cura_verify_text		
Description			
Total	1		
Passed	0	Failed	1
Error	0	Incomplete	0
Start	2021-07-28 21:13:27	End	2021-07-28 21:14:09
Elapsed	41.577s		

#	ID	Description	Status
1	Test Cases/Verify_Text_TC		FAILED

Lab 12: Execution profile & Exception Handling in Katalon

Execution Profile

Execution Profile helps cover multiple and different environments to execute your automation test scripts with ease. You can configure the testing environment in terms of data and behaviors through Global variables. Mainly there are three types of variables supported in Katalon Studio, Groovy Variable, Test Case Variable(A Test Case Variable can be used to parameterize a test case or to call that test case with different inputs.) and Global Variable (Execution Profiles) A Global Variable can be accessed anywhere inside your project. **Groovy Variables**

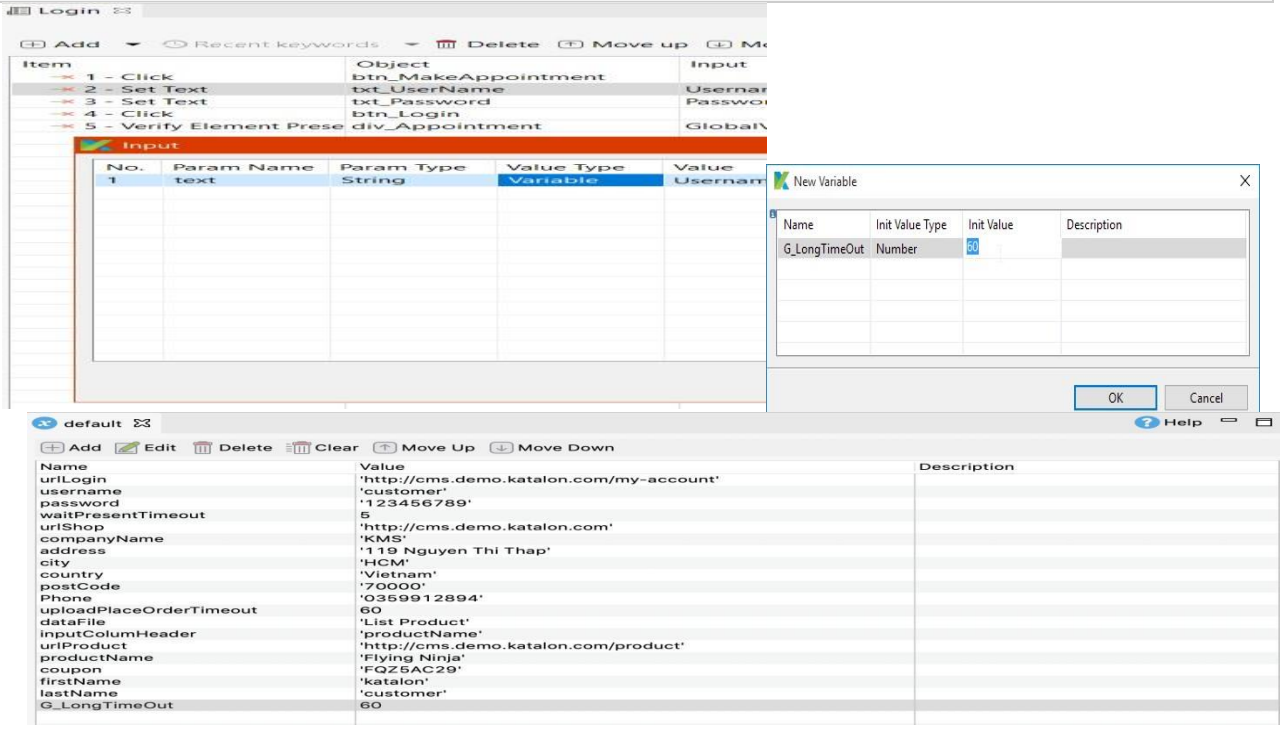



```
// x is defined as a variable of String type
String x = "Hello";
// y is defined as a variable of int type
int y = 5;
// The value of the variables are printed to the
console println(x); println(y);
/* O/p will be printed on Console */
```

Test Case Variables: You can manage Test Case Variable in the Variables tab of the Test Case Editor.To add variable using grid view, switch to Variables tab of your Test Case. Then click Add. A new row is added to the variable list. Modify the variable details and save the test case once done. Test Case Variables can be referred in test case as Groovy variables, e.g.

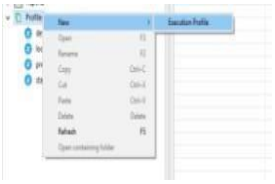
```
import static com.kms.katalon.core.testobject.ObjectRepository.findTestObject

import com.kms.katalon.core.webui.keyword.WebUiBuiltInKeywords as WebUI import

internal.GlobalVariable as GlobalVariable

WebUI.click(findTestObject('Page_CuraHomepage/btn_MakeAppointment'))

WebUI.setText(findTestObject('Page_Login/txt_UserName'), Username)

WebUI.setText(findTestObject('Page_Login/txt_Password'), Password)

WebUI.click(findTestObject('Page_Login/btn_Login'))
```

```
landingPage =
WebUI.verifyElementPresent(findTestObject('Page_CuraAppointment/div_Appointme
nt'), GlobalVariable.G_Timeout)
```



Global Variables: A **global variable** is a variable defined in the execution profile and can be used in a test case, test object, web service object, and email configuration in a project.

Create a profile



Like other test artifacts, you can CRUD the **Execution Profile** in the **Tests Explorer**.

You need to define a profile's content by adding Global variables. Do as follows:

1. Select a profile > click **Add**.
2. In the **New Variable** dialog, specify details for the variable > click **OK**

3. The variable is added to the profile accordingly.

View a profile

Execution profile is provided with **Manual view** and **Script view**. In the Script view, an XML editor is available for adding variables via script. Depending on the project needs, you can create as many profiles as you want to.

In the **Script view**, profiles are in sync once a similar list of **Global Variables** is required for testing different environment types. To conduct, copy and paste the variables list from one profile to another.

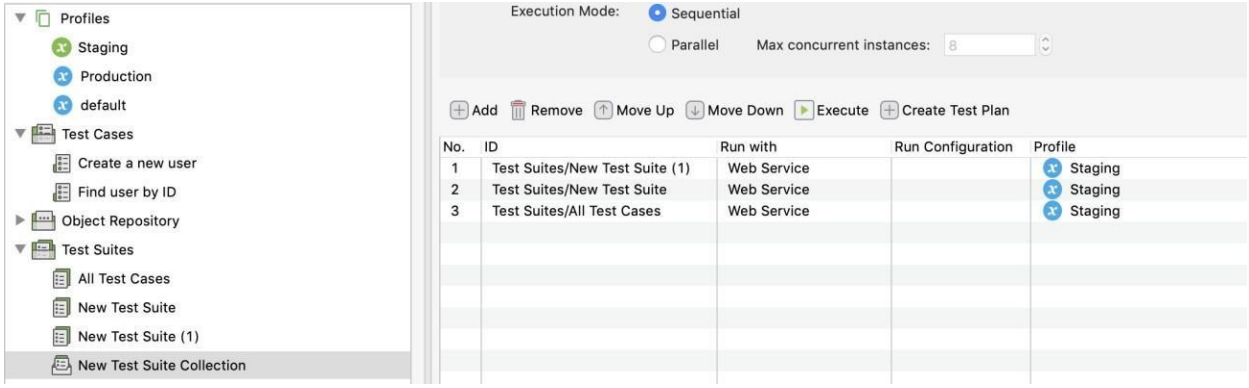
Set default profile at project level

A default profile is considered as a place to comprise commonly used global variables. Other profiles can either inherit or override the global variables stored in the default one.

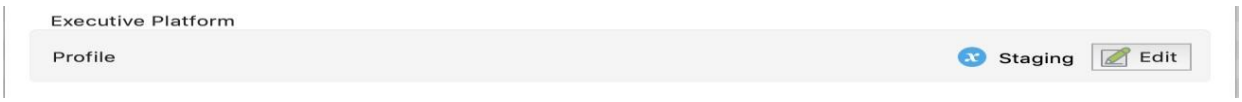
You may have multiple profiles for executing your tests, for instance, *staging* and *production* profiles. It would be convenient to set a profile as your default one in every execution of a project.

Right-click on your desired execution profile and select **Set as default Execution Profile**.

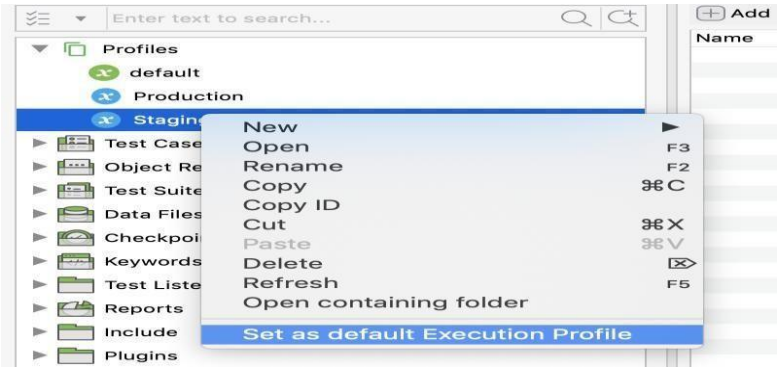
This profile becomes a default execution option for Test Case, Test Suite, and Test Suite Collection.



It's also applied for the Command Generator's Executive Platform in case you use Katalon Runtime Engine.



Profile Inheritance: reduces your effort spent on modification and recreation of the same global variables in derived profiles.



Suppose Katalon Studio does not find variables used in the test within the designated profile (any profiles but default). In that case, it will look into the default profile and use its variables to execute the test. **How to utilize Profile Inheritance**

Commonly used global variables should be stored in the **default** profile and should store other sets of global variables in the **derived** (custom) profiles to avoid duplicated code and for better management.

Examples: The following examples illustrate how the **Profile Inheritance** feature works.

- Given the following test case:

default production stagging sample

Add Recent keywords Delete Move up Move down Edit tags Set default view

Item	Object	Input
1 - Method Call Statement		println("name: " + GlobalVariable.name)
2 - Method Call Statement		println("usage: " + GlobalVariable.usage)
3 - Method Call Statement		println("reference: " + GlobalVariable.reference)
4 - Method Call Statement		println("running with: " + GlobalVariable.serverURL)
5 - Method Call Statement		println("credential: " + GlobalVariable.credential)

Execute default profile with the given test case:

default production stagging sample

Add Edit Delete Clear Move Up Move Down

Name	Value
name	'default profile'
usage	'show how inheritance, overiding and polymorphism work'
reference	'https://docs.katalon.com/katalon-studio/docs/execution-profile...
serverURL	'http://localhost:8080'
credential	''

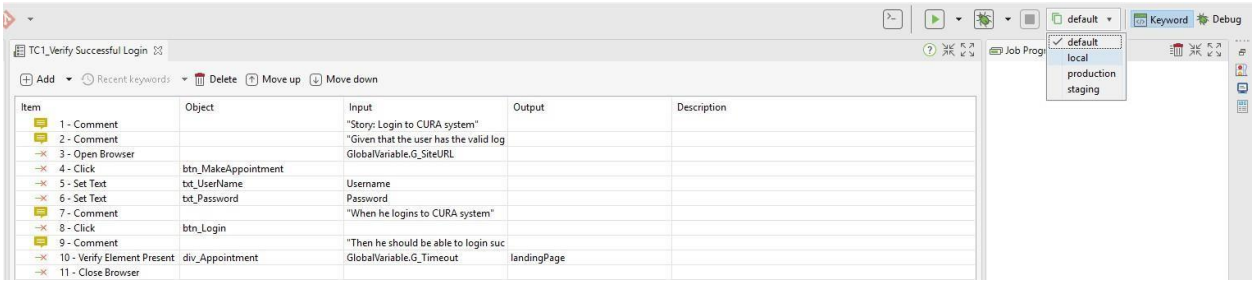
The result is shown as below:

```
Problems  Event Log  Console  Log Viewer  Self-healing Insights
<terminated> TempTestCase1614264711288 [Katalon] /Applications/Katalon Studio.app/Contents/Eclipse/jre/Contents/Home/bin/java  (Feb 25, 202
2021-02-25 21:51:57.971 INFO  c.k.katalon.core.main.TestCaseExecutor   - --------------------
2021-02-25 21:51:57.974 INFO  c.k.katalon.core.main.TestCaseExecutor   - START Test Cases/sample
2021-02-25 21:51:59.714 DEBUG testcase.sample                          - 1: println("name: " + name)
name: default profile
2021-02-25 21:52:00.321 DEBUG testcase.sample                          - 2: println("usage: " + usage)
usage: show how inheritance, overiding and polymorphism work
2021-02-25 21:52:00.357 DEBUG testcase.sample                          - 3: println("reference: " + reference)
reference: https://docs.katalon.com/katalon-studio/docs/execution-profile-v54.html#inheritance-profile
2021-02-25 21:52:00.486 DEBUG testcase.sample                          - 4: println("running with: " + serverURL)
running with: http://localhost:8080
2021-02-25 21:52:00.500 DEBUG testcase.sample                          - 5: println("credential: " + credential)
credential:
2021-02-25 21:52:00.591 INFO  c.k.katalon.core.main.TestCaseExecutor   - END Test Cases/sample
```

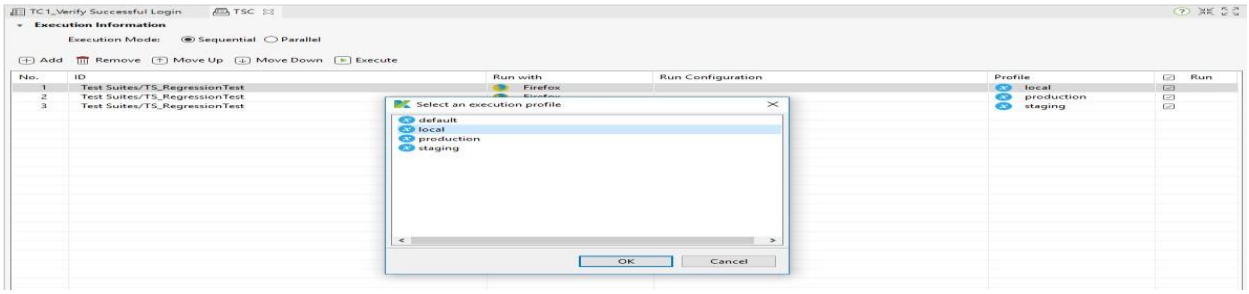
Use a profile: By default, Katalon Studio uses the **default** profile for executing tests, as indicated on the screen's top right corner. You can select any available execution profiles in the drop-down menu.

The following section shows you a usage example. There are three profiles based on testing environments: **local**, **staging**, and **production**.

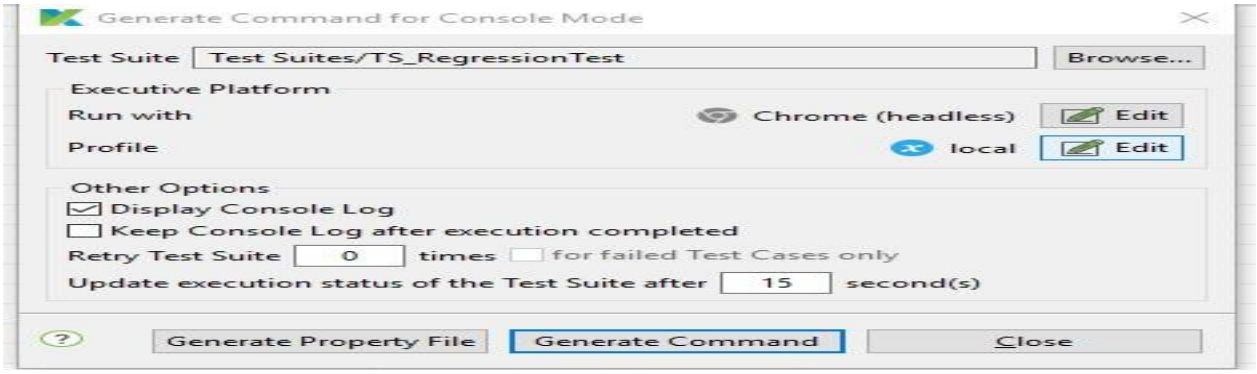
- **For test cases or test suites**: Select your desired profile on the top right > **all Global Variables** within your current project automatically uses these values.



For Test Suite Collection: Select your desired profile to be executed with your Test Suite on the **Profile** column.



For Console Mode execution: Select your desired profile on the **Profile** field.



The **Generated Command** has **executionProfile** parameter so that you can change it manually. For example:

```
katalonc -noSplash -runMode=console -consoleLog -projectPath="C:\Users\Admin\Katalon Studio\yourProject.prj" -retry=0 -testSuitePath="Test Suites/TS_RegressionTest" -executionProfile="local" -browserType="Chrome (headless)
```

Global variables are Test Suite scoped. If any Global Variable value is changed during runtime, this change will not be shared among Test Suites. This infers that you can modify Global Variables when executing a particular Test Suite without affecting other Test Suites' Global Variables in the Test Suite Collection.

You can find "Test Suites/New Test Suite (1)" is listed twice in the following screenshot. The first one uses "default," and the second one has "stagging". This association proves that **a Profile is Test Suite scoped**. Otherwise, the association can not be logically valid.

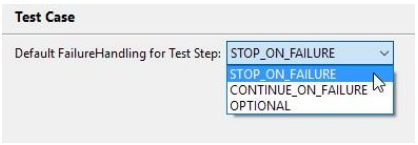
Use a Global Variable: Any test cases across a project can use global variables - for example, input data for keywords in Manual View (highlighted in blue) or params when binding Data for Test Execution (highlighted in red).

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint

CustomKeywords.'sample.Login.loginIntoApplicationWithGlobalVariable'()

WebUI.waitForElementPresent(findTestObject('Pages/Shop page/lnkShop'), GlobalVariable.waitPresentTimeout)

WebUI.click(findTestObject('Pages/Shop page/lnkShop'))

TestData product = findTestData(GlobalVariable.dataFile)
List<String> productList = product.getAllData().stream()
                           .map{data -> data[0]}/*get first column of each row in data file */
                           .collect(Collectors.toList())/*add collect and parse to list*/

for(def productName : productList){
    CustomKeywords.'sample.Shop.addToCart'(productName.toString(), GlobalVariable.urlProduct)
}
CustomKeywords.'sample.Checkout.CheckoutShopWithGlobalVariable'()
WebUI.closeBrowser()
```

Failure Handling : Failure handling settings allow users to decide whether Katalon Studio will continue running or not in case of errors occurs during execution. Currently, Katalon Studio supports the following failure handling options:

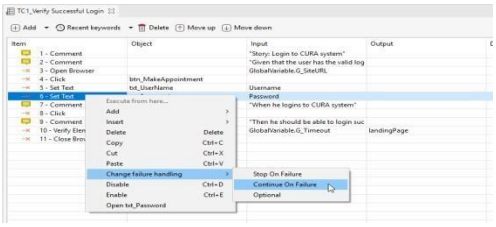
Option	Description
Stop on Failure	Katalon Studio will stop execution should there be any error occurs.The step with errors will have **Failed** status.
Continue on Failure	Katalon Studio will continue in spite of any error during its execution.The step with errors will have **Failed** status.
	Katalon Studio will continue in spite of any error during its execution.The step with errors will have **Warning** status.

Default failure handling behavior: Follow these steps to define the default behavior for failure handling to be applied across your project:



1. From Katalon Studio menu, access **Project > Settings > Test Design > Test Case**.
2. Select the preferred option for the default behavior of **Failure Handling**. Click **OK** when you're done.

Override failure handling behavior You can override the default failure handling behavior for each test step manually in either **Manual view** or **Scripting view** of test case. **In Manual View**

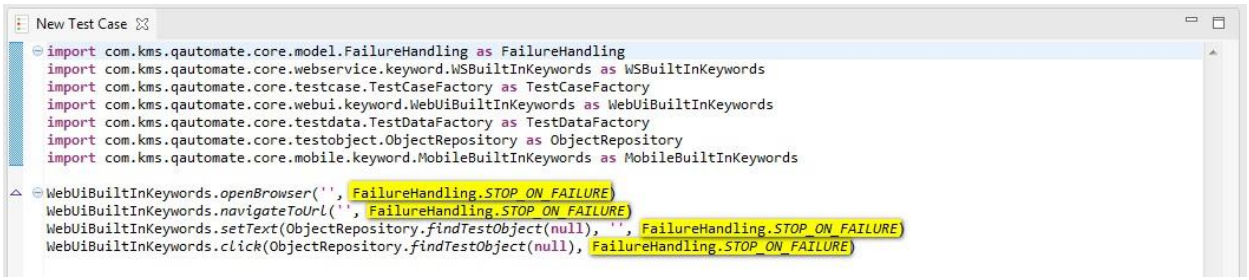


1. Right click on the step that you want to change the failure handling behavior to trigger its Right click on the step that you want to change the failure handling behavior to trigger its context menu
2. Select the preferred failure handling option and save your test case.

In Scripting View: For all built-in keywords in Katalon Studio, you can add *FailureHandling* as the last parameter.When editing a keyword in Scripting mode, use any of these option to specify its behavior.

```
FailureHandling.STOP_ON_FAILURE
FailureHandling.CONTINUE_ON_FAILURE
FailureHandling.OPTIONAL
```

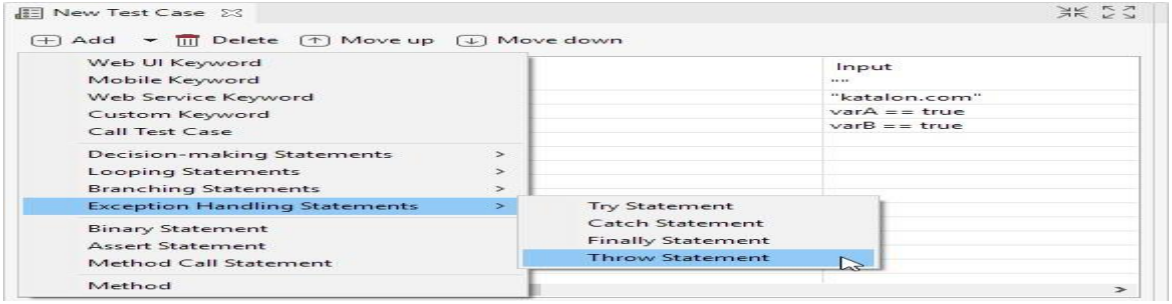
For example:



Exception handling block

In Manual view

Open a test case in **Manual** view, then navigate to **Exception Handling Statements** from command toolbar.



Refer to following table for the usage of each statement:

Statement	Description
Try	This statement indicates that all steps within will be monitored by **exception handlers**.
Throw	Before you can **Catch** an exception, some code must throw one. Regardless of what throws the exception, it's always involved with the **Throw** statement
Catch	Katalon Studio will execute all steps within when there is any issue occurred during execution of the **Try** block.
Finally	This is the last part of the **Try-Catch-Finally** structure and all steps within this will be executed regardless of any exception.

In Script view: The **Script** view of test cases allows you to programmatically define and handle exception easily using either Groovy or Java language.

```
try {
    WebUI.openBrowser('')

    WebUI.navigateToUrl('katalon.com')

    if (WebUI.getText(findTestObject('Object Repository/txt_signin')).length() < 0){
        throw new com.kms.katalon.core.exception.StepErrorException('Value required!')
    }

}
catch (StepErrorException e) {
    this.println(e)
}
catch (Exception e) {
    this.println("General issue occurs.")
}
finally {
    this.println("Navigate to a page.")
}
```

EXERCISE:

a. Test Cura healthcare to book an appointment ,An exception should throw when the length of Username is less or equal to 8.

b. Script

book appointment

```
WebUI.openBrowser('')

def username1 = 'John Doe'

def password1 = 'ThisIsNotAPassword'

WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com/')

WebUI.verifyElementText(findTestObject('Object Repository/Page_CURA Healthcare Service/h3_We Care About Your Health'), 'We Car

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Make Appointment'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'), username1)

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Password_password'), password1)

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Login'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/label_Apply for hospital readmission'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Visit Date (Required)_visit_date'))

WebUI.click(findTestObject('Page_CURA Healthcare Service/td_28'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/textarea_Comment_comment'), 'Appointment')

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Book Appointment'))

WebUI.closeBrowser()
```

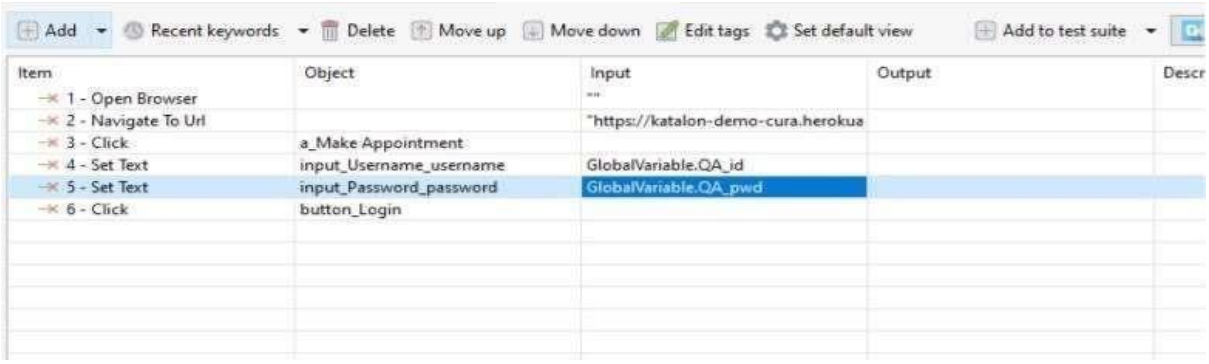
c. Declare a test case variable inside “Login Test” and execute.

d. Script:

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint

WebUI.openBrowser('')

WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com/')

WebUI.verifyElementText(findTestObject('Object Repository/Page_CURA Healthcare Service/h3_We Care About Your Health'), 'We Care Ab

WebUI.rightClick(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Make Appointment'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Make Appointment'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Demo account_form-control'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'), username)

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Demo account_form-control_1'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Password_password'), password)

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Login'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Apply for hospital readmission_hospit_63901f'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/label_Medicare'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/span_Visit Date (Required)_glyphicon glyphi_cada34'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/td_20'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Apply for hospital readmission_hospit_63901f'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/label_Medicare'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/span_Visit Date (Required)_glyphicon glyphi_cada34'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/td_20'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/textarea_Comment_comment'), 'hello')

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Book Appointment'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Go to Homepage'))

WebUI.closeBrowser()
```

e. Declare a global variable in QA profile and test "Login feature".

e. Test Cases:



Script

```
WebUI.openBrowser('')

WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com/')

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Make Appointment'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'), GlobalVariable.QA_id)

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Password_password'), GlobalVariable.QA_pwd)

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Login'))
```

g. Test Cura healthcare to book an appointment, An exception should be thrown when the length of username is less or equal to 8

h. Script:

```
WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com/')

try {
    WebUI.verifyElementText(findTestObject('Object Repository/Page_CURA Healthcare Service/h3_We Care About Your Health'),
        'Health')
}
catch (Exception e) {
    println(e)
}

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Make Appointment'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'), GlobalVariable.UID)

try {
    def data = WebUI.getText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'))

    if (data.length()<= 8) {
        println('should be greater than 8')
    }
}
catch (Exception e) {
    println(e)
}

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Password_password'), GlobalVariable.Pwd)

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Login'))
```

i. Verify the date (selected) is not the previous date ,if so an exception will print on console. Hint: Use conditional control statements/Method call statements if required /*Attach all printout here including test cases, scripts, test summary reports

Script:

```
WebUI.openBrowser('')

WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com/')

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/a_Make Appointment'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Demo account_form-control'))

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/div_Demo account_input-group'))

WebUI.setText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Username_username'), 'John Doe')

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/div_Demo account_input-group_1'))

WebUI.setEncryptedText(findTestObject('Object Repository/Page_CURA Healthcare Service/input_Password_password'), 'g3/DOGG74jC3Flrr

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Login'))

WebUI.setText(findTestObject
    ('Object Repository/Page_CURA Healthcare Service/input_Visit Date (Required)_visit_date'), '26/06/2021')

if(WebUI.getText(findTestObject
    ('Object Repository/Page_CURA Healthcare Service/input_Visit Date (Required)_visit_date'))<='26/06/2021'){
    println("Date should not be previous")
}

WebUI.click(findTestObject('Object Repository/Page_CURA Healthcare Service/button_Book Appointment'))
```

Test Summary Report:

Lab 12

Execution Environment

Host name	My PC - DESKTOP
OS	Windows 10 64bit
Katalon version	8.0.5.208
Browser	Chrome 91.0.4215.127

Summary

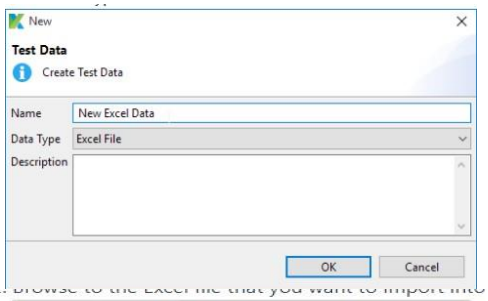
ID	Test Suites/Lab 12		
Description			
Total	5		
Passed	5	Failed	0
Error	0	Incomplete	0
Start	2021-07-29 10:22:23	End	2021-07-29 10:24:03
Elapsed	1m - 40.600s		

#	ID	Description	Status
1	Test Cases/book appointment		PASSED
2	Test Cases/testcase1		PASSED
3	Test Cases/globalVariable		PASSED
4	Test Cases/exceptionHandling		PASSED

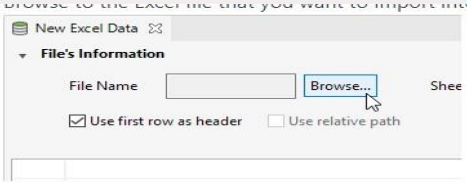
Lab 13: Data Driven Testing

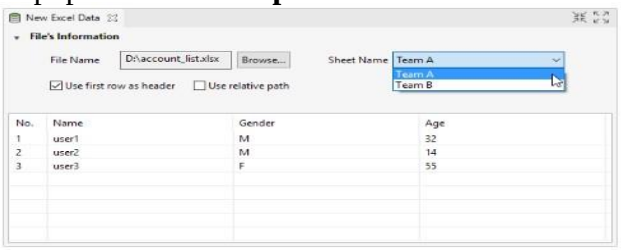
Create an Excel Test Data

1. Select **File > New > Test Data** from the main menu. The **New Test Data** dialog is displayed. Enter the name for your test data and select **Data Type** as **Excel File**. Click **OK**.



2. Browse to the Excel file that you want to import into Katalon Studio. Data from the selected Excel file is populated into the **preview section** below.

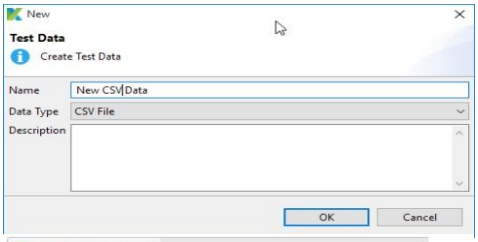




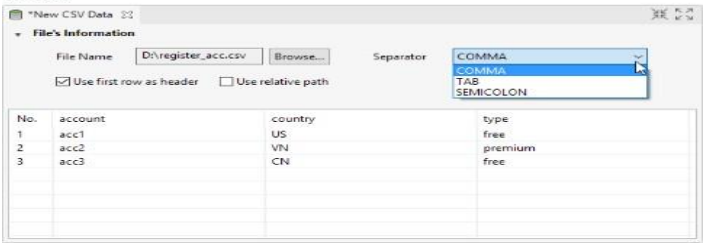
3. Save the Test Data when you finish. The data set defined here can be utilized in other configurations. For example, you can use it to input data for keywords in Manual View or data for Test Execution when setting up test suites.

Create a CSV Test Data

1. Select **File > New > Test Data** from the main menu. The **New Test Data** dialog is displayed. Enter the name for your test data and select **Data Type** as **CSV File**. Click **OK**.



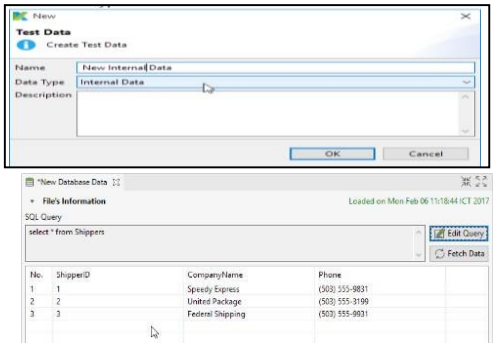
2. Browse to the CSV file that you want to import into Katalon Studio.Data from the selected CSV file is populated to the **preview section** below.



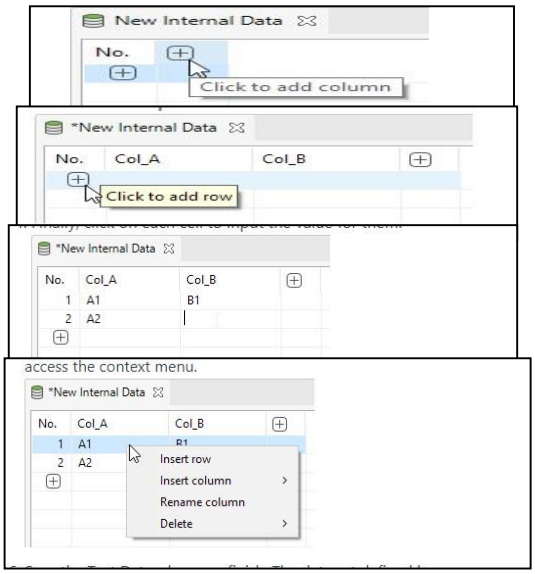
3 .Save the Test Data when you finish. The data set defined here can be utilized in other configurations. For example, you can use it to input data for keywords in Manual View or data for Test Execution when setting up test suites.

Create an Internal Test Data

With **Internal Data**, you can freely define the data in tabular format. It's up to you to decide how many columns, rows, and what value for each cell.

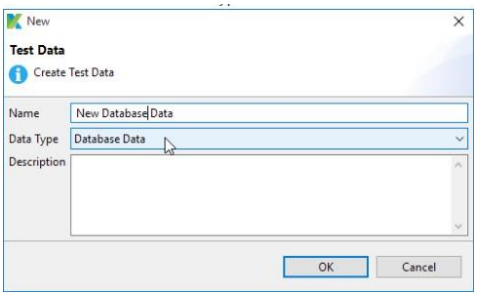


1. Select **File > New > Test Data** from the main menu. The **New Test Data** dialog is displayed. Enter the name for your test data and select **Data Type** as **Internal Data**. Click **OK**.
2. In the Editor View, select the option to add a new column.
3. Select the option to add a new row.
4. Finally, click on each cell to input the value for them.
5. You can edit or delete the columns (or rows) by right-clicking to access the context menu.
6. Save the Test Data when you finish. The data set defined here can be utilized in other configurations. For example, you can use it to input data for keywords in Manual View or data for Test Execution when setting up test suites.
7. Create a Database Data

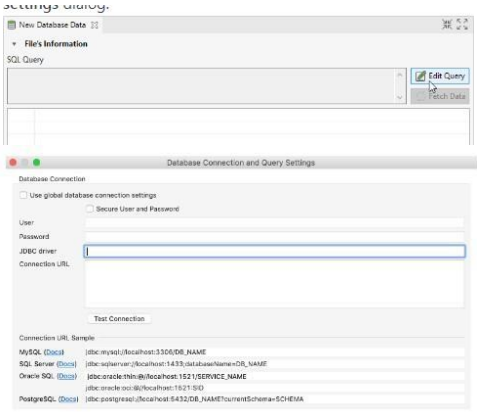


With **Internal Data**, you can freely define the data in tabular format. It's up to you to decide how many columns, rows, and what value for each cell.

1. Select **File > New > Test Data** from the main menu. The **New Test Data** dialog is displayed. Enter the name for your test data and select **Database Data** as **Data Type**. Click **OK**.
2. Click **Edit Query** to open the **Database Connection and Query settings** dialog.
3. Enter the connection details as well as the data query then click **OK**.Starting from **Katalon Studio version 7.0.0 and later**, you can query data from additional database sources with the **JDBC Driver** field in the dialog. You can enter the ClassDriverName of the database that has a library support connection (JDBC).

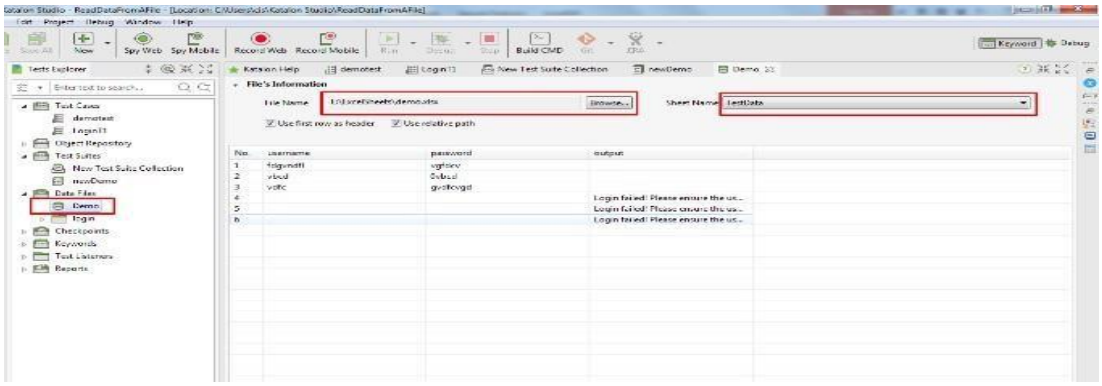


4. Queried data is fetched and loaded respectively into the **preview section** .
5. Save the Test Data when you finish. The data set defined here can be utilized in other configurations. For example, you can use it to input data for keywords in Manual View or data for Test Execution when setting up test suites.



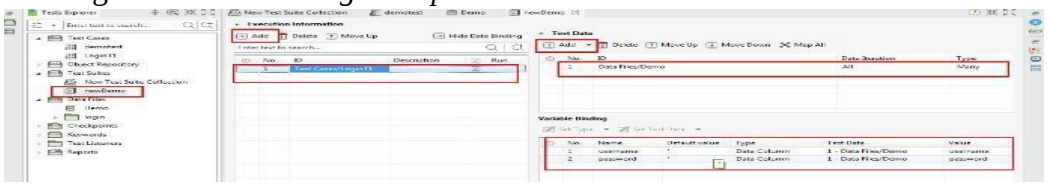
How to read Data from Excel File in Katalon:

Step 1: create a new data file(excel) example "demo.xlsx". Step 2: And enter some data into the file.
Step 3: Create a new data file in katalon studio.
Step 4: And browse your external file and select the sheet.

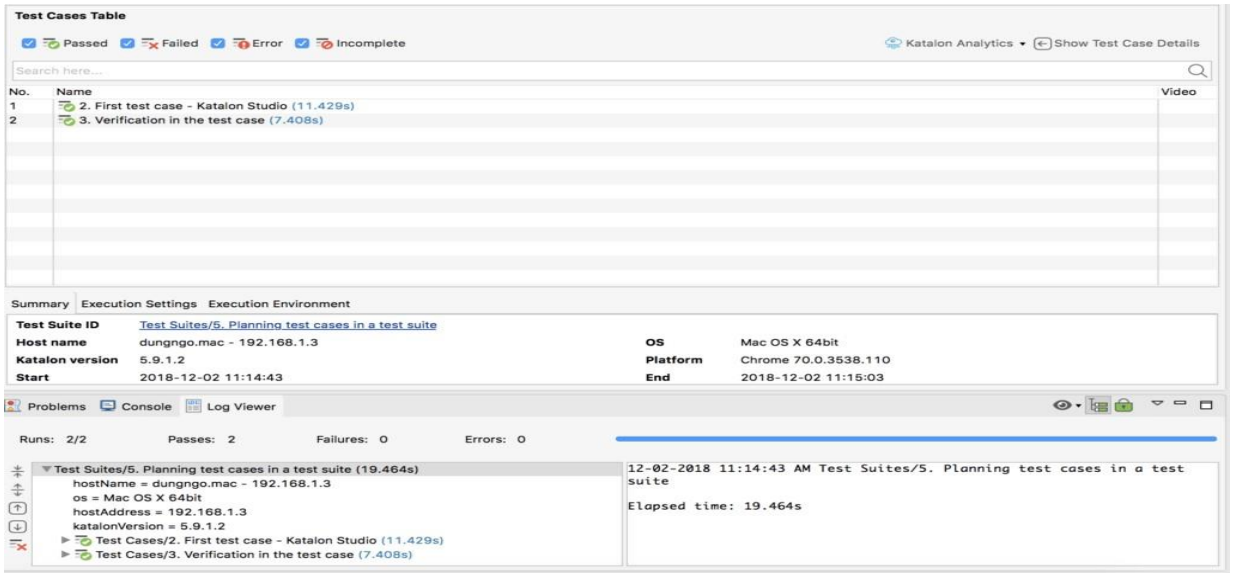


Step 5: Create test case and **define some variable in the test case**.

Step 6: Now create a TestSuite and configure a test case. And bind the variables with the column name using *Show data binding→MapAll*.



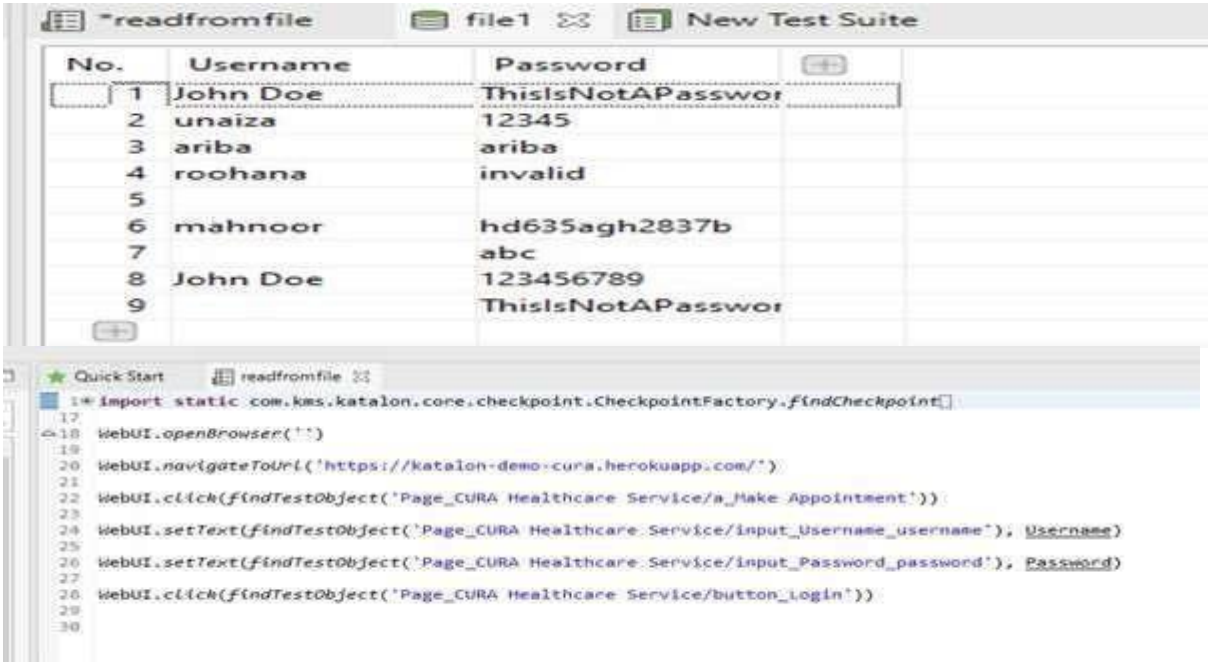
Step9. Click save & Run to see how it works.



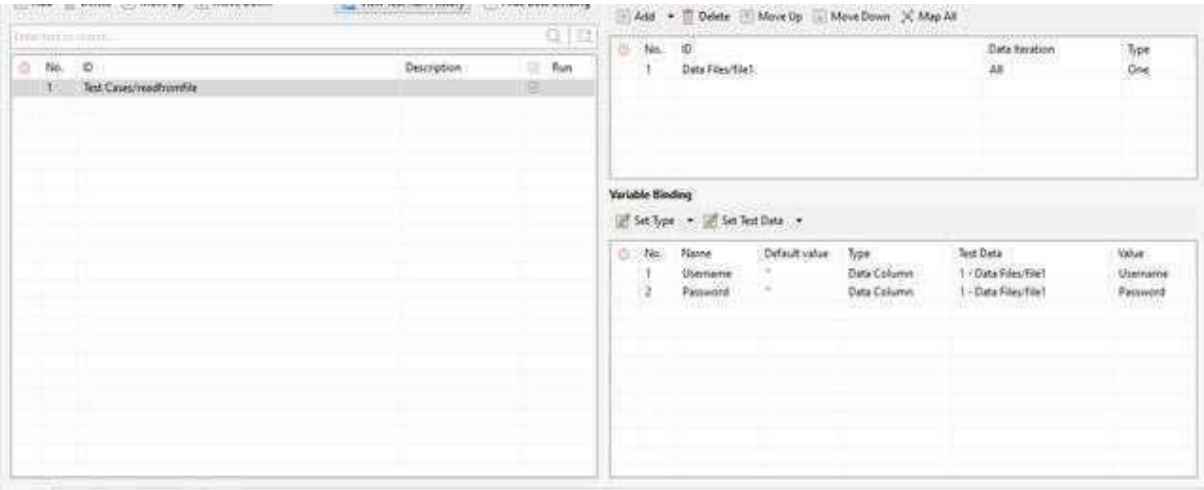
EXERCISE:

a. Create an internal data file (having Username and password) ,Test CURA HealthCare Website on all possible combinations of userid & password. Generate and attach Test report/*Attach all screenshot/printout here*/ Also Attach Test suit performance summary through TestOp feature.

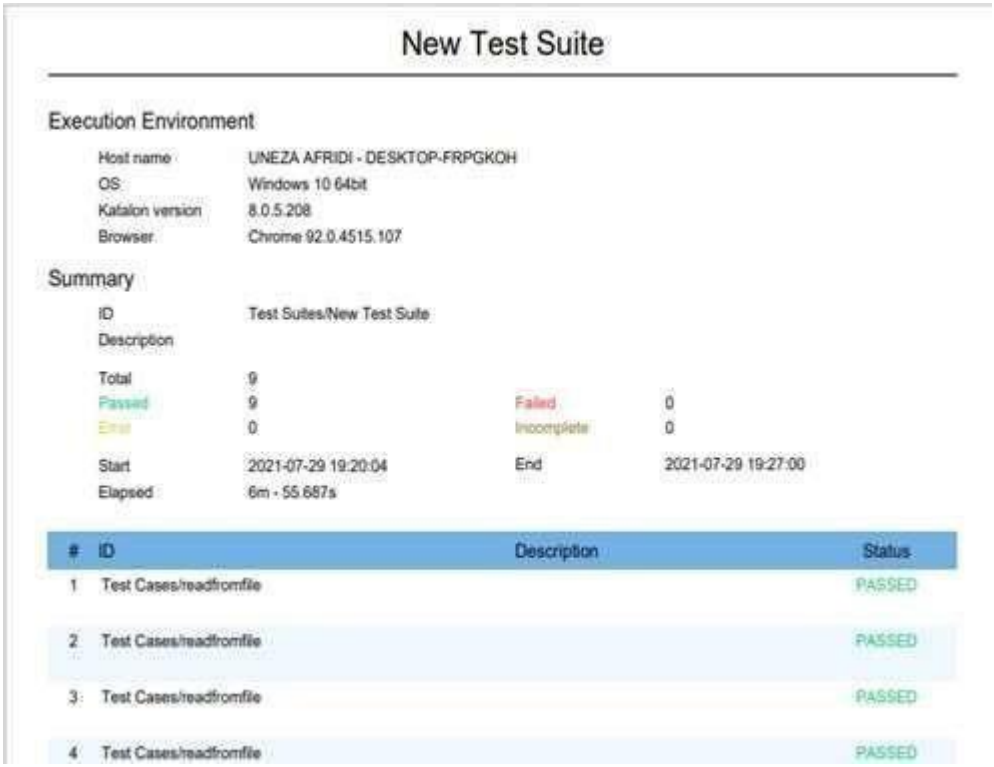
AN INTERNAL DATA FILE WITH USERNAME AND PASSWORD:



TEST SUITE WITH DATA BINDING:



TEST SUMMARY REPORT:



New Test Suite

Execution Environment

Host name	UNEZA AFRIDI - DESKTOP-FRPGKOH
OS	Windows 10 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

Summary

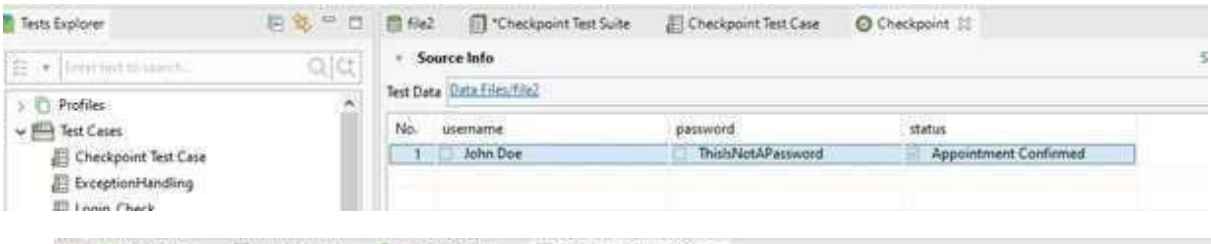
ID	Test Suites/New Test Suite		
Description			
Total	9		
Passed	9	Failed	0
Error	0	Incomplete	0
Start	2021-07-29 19:20:04	End	2021-07-29 19:27:00
Elapsed	6m - 55.687s		

#	ID	Description	Status
1	Test Cases/readfromfile		PASSED
2	Test Cases/readfromfile		PASSED
3	Test Cases/readfromfile		PASSED
4	Test Cases/readfromfile		PASSED

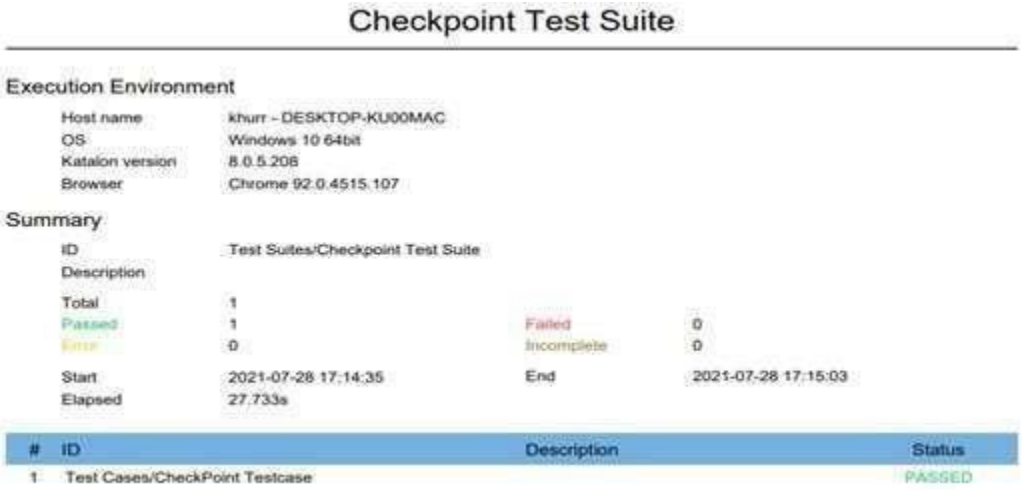
Report Katalon Testops:

b. Add checkpoint in the internal data file [refer part a here](add one more column named"status") and execute test case to validate on the basis of "status" whether appointment is confirmed or cancel /*Attach All screenshot/printout here*/ Hint : rightclick to Checkpoint à New

CHECKPOINT:



REPORT:



Checkpoint Test Suite

Execution Environment

Host name	khurr - DESKTOP-KU00MAC
OS	Windows 10 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

Summary

ID	Test Suites/Checkpoint Test Suite		
Description			
Total	1		
Passed	1	Failed	0
Error	0	Incomplete	0
Start	2021-07-28 17:14:35	End	2021-07-28 17:15:03
Elapsed	27.733s		

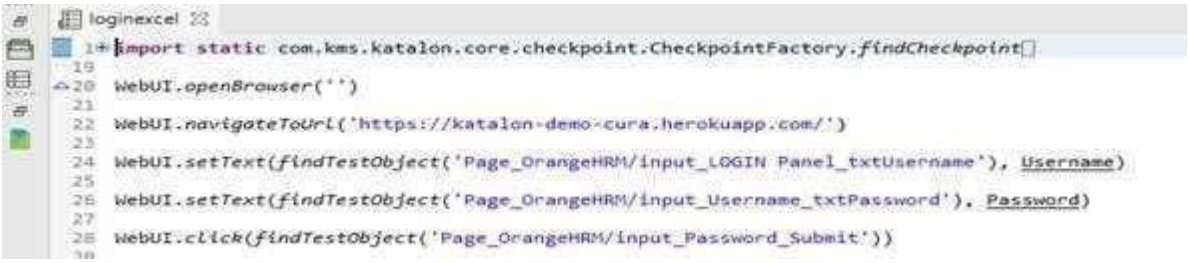
#	ID	Description	Status
1	Test Cases/CheckPoint Testcase		PASSED

c. Attach an excel sheet ”Testdata”(having Username and password, all possible combinations), Create and execute testcases for Orangehrm ;generate and attach Testreport/*Attach All screenshot/printout here*/

EXCEL FILE:

	A	B	C	D
1	Username	Password		
2				
3	Admin	admin123		
4	admin	admin123		
5		admin123		
6	Admin			
7	xyz	xyz		
8	123	admin123		
9	Adm	admin123		
10				

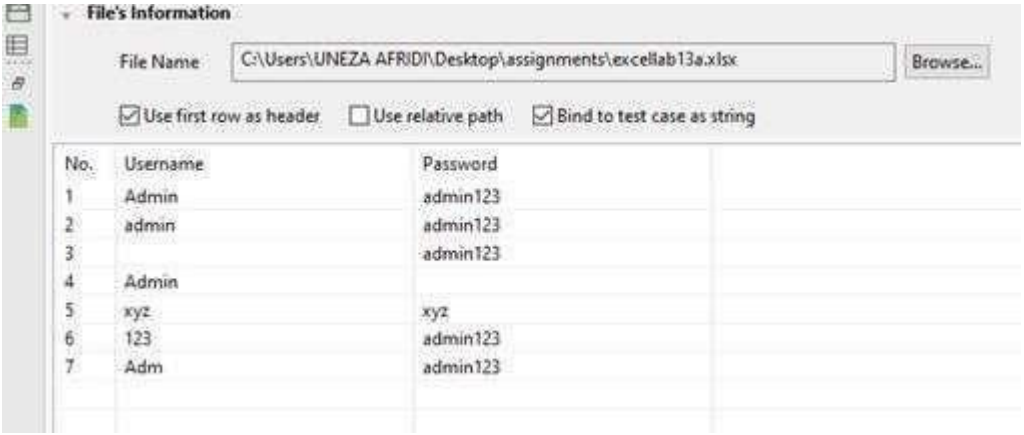
TEST CASE FOR READING EXCEL FILE:



loginexcel

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint
19
20 WebUI.openBrowser('')
21
22 WebUI.navigateToUrl('https://katalon-demo-cura.herokuapp.com/')
23
24 WebUI.setText(findTestObject('Page_OrangeHRM/input_LOGIN Panel_txtUsername'), Username)
25
26 WebUI.setText(findTestObject('Page_OrangeHRM/input_Username_txtPassword'), Password)
27
28 WebUI.click(findTestObject('Page_OrangeHRM/input_Password_Submit'))
```

IMPORT EXCEL FILE IN PROJECT:



REPORT:

d. Create another column in (above) excel sheet named “Status” through test case write “passed” or “fail” in status column during Execution. /* use conditional control statements and exception handling if required*/ Attach All printout here

EDITING EXCEL FILE DURING EXECUTION:

```
public class WriteExcel {
    @Keyword
    public void demoWriteExcel (String strTest) throws IOException
    {
        FileInputStream file = new FileInputStream ('G:\\SEMESTER 6\\SQE lab\\Testdata.xlsx');
        XSSFWorkbook workbook = new XSSFWorkbook(file);
        XSSFSheet sheet = workbook.getSheet('Sheet1');
        int x = GlobalVariable.i
        println('strTest: '+ strTest)
        println('x: '+x)
        XSSFRow row = sheet.getRow(x)
        int colNum = row.getLastCellNum()
        XSSFCell cell = null
        if (cell==null)
            cell = row.createCell(colNum)
        cell.setCellValue(strTest)
        FileOutputStream outFile =new FileOutputStream('G:\\SEMESTER 6\\SQE lab\\Testdata.xlsx');
        workbook.write(outFile);
        outFile.close()
        GlobalVariable.i=x+1
        println("GlobalVariable.i: " + GlobalVariable.i)
            }

        }
```

EXCEL FILE AFTER EXECUTION:

	A	B	C	D
1	Username	Password	Status	
2	Admin	admin123	failled	
3	admin	admin123	failled	
4		admin123	failled	
5	Admin		failled	
6	xyz	xyz	failled	
7	123	admin123	failled	
8	Adm	admin123	failled	
9				

REPORT:

New Test Suite

Execution Environment

Host name	UNEZA AFRIDI - DESKTOP-FRPGKOH
OS	Windows 10 64bit
Katalon version	8.0.5.208
Browser	Chrome 92.0.4515.107

Summary

ID	Test Suites/New Test Suite		
Description			
Total	7		
Passed	0	Failed	7
Error	0	Incomplete	0
Start	2021-07-30 00:26:21	End	2021-07-30 00:31:46
Elapsed	5m - 25.645s		

#	ID	Description	Status
1	Test Cases/loginexcel		FAILED
2	Test Cases/loginexcel		FAILED
3	Test Cases/loginexcel		FAILED
4	Test Cases/loginexcel		FAILED
5	Test Cases/loginexcel		FAILED
6	Test Cases/loginexcel		FAILED
7	Test Cases/loginexcel		FAILED

Lab 14: Desktop App Testing

Katalon Studio fully supports automation test for desktop apps written in the following platforms: Universal Windows Platform (UWP), Windows Forms (WinForms), Windows Presentation Foundation (WPF), and Classic Windows (Win32).

Set up Windows Application Driver on a local Windows 10 machine

You must need to install and run Windows Application Driver on a Windows 10 machine before performing automation test for Windows application. You can install WinAppDriver on a local Windows 10 machine in two ways:

- Open this folder C:\Program Files (x86)\Windows Application Driver, then double-click on WinAppDriver.exe file; or

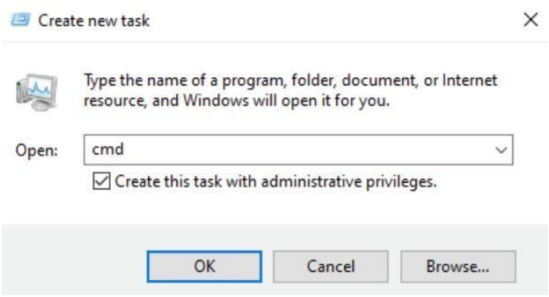
- From the Katalon Studio toolbar, select Tools > Windows > Install WinAppDrivers. The Windows Application Driver Setup window will pop up. Follow the instructions to install the Windows Application Driver. Then run WinAppDriver.exe.

Set up Windows Application Driver on a remote Windows 10 machine

1. Run the installer on a Windows 10 machine where your application under test is installed and will be tested.
2. Enable Developer Mode in Windows settings.
3. Run WinAppDriver.exe from the installation directory (E.g. C:\Program Files (x86)\Windows Application Driver)

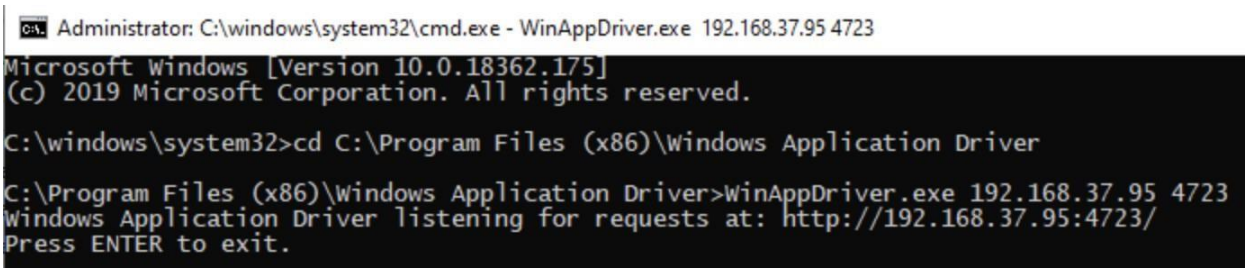
On the test machine:

1. Open Task Manager > Select File > Create a new task.
2. Select Create this task with administrative privileges and enter cmd in the Open text box, then click the OK button.

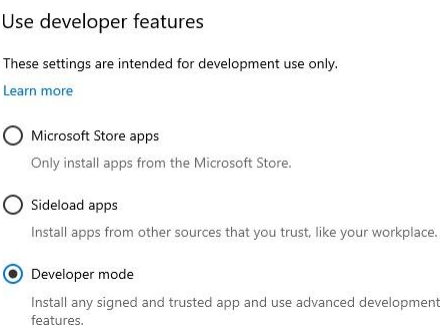


3. Run cd to folder C:\Program Files (x86)\Windows Application Driver.

4. Run WinAppDriver.exe <IP_Adress> <Port>.
   - *IP_Adress* is the public IP address of the test machine. Run ipconfig.exe to determine the IP address.
   - *Port* is the public port of the remote machine. By default, it should be 4723.

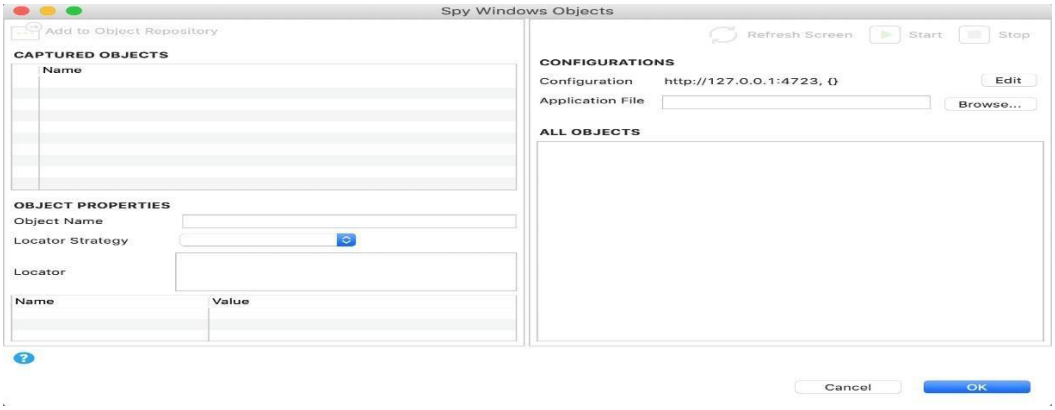


5. Enable Developer Mode on the test machine



Spying Windows objects: To start spying a Windows object, click Spy Windows Object icon on the main toolbar of Katalon Studio.

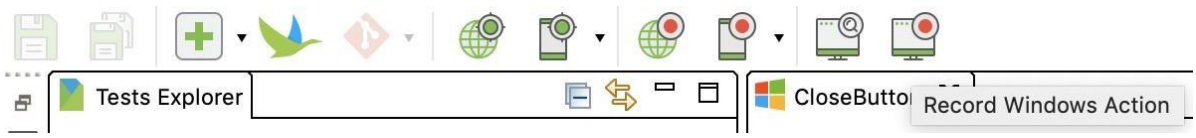
Configurations: In the CONFIGURATIONS section:

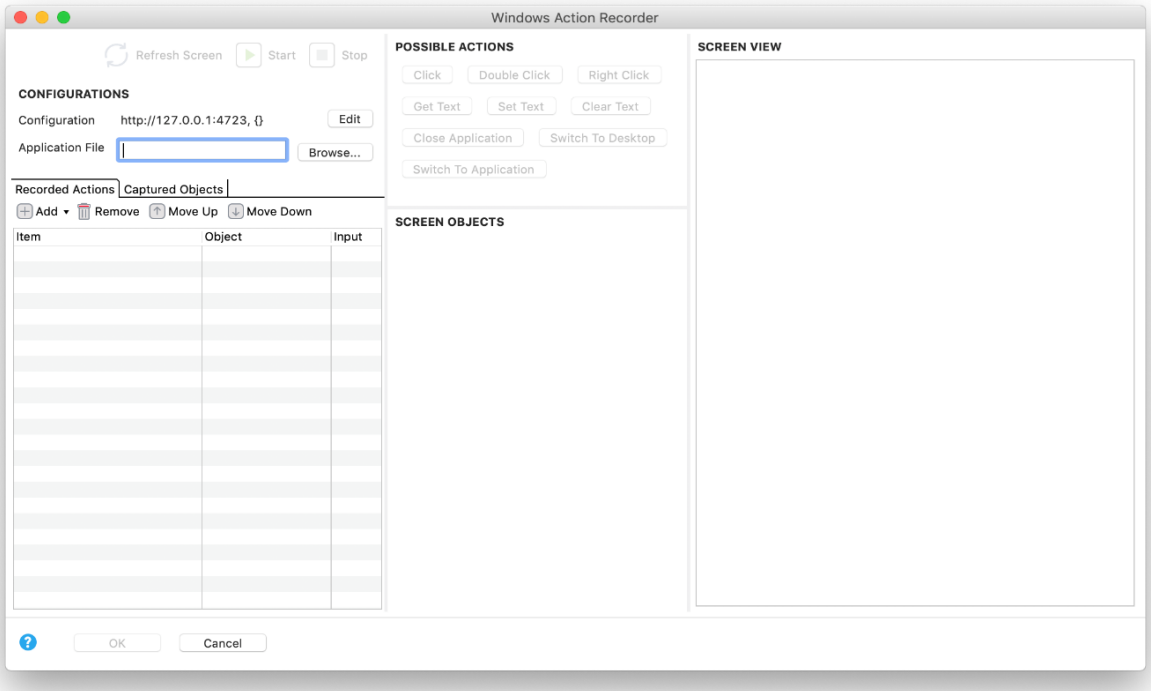


- Configuration: where you can view and edit the WinAppDriver URL and Desired Capabilities.
- Application File: the absolute path to the Windows Executable File (*.exe) of the testing machine. For Windows users, click Browse... button to locate the application file.
- The Start button is to be enabled after the Application File text box is filled.

When the application starts, Katalon Studio starts capturing all available Windows objects on the current screen of the testing machine and shows them at the ALL OBJECTS section.

To start recording a Windows action, click Record Windows Action icon on the main toolbar of Katalon Studio.





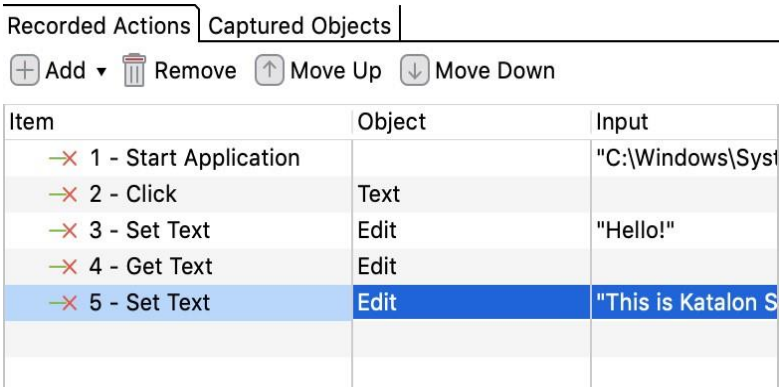
Action Bar: The Action Bar located on the top left corner contains the Refresh

screen, Start and Stop buttons.

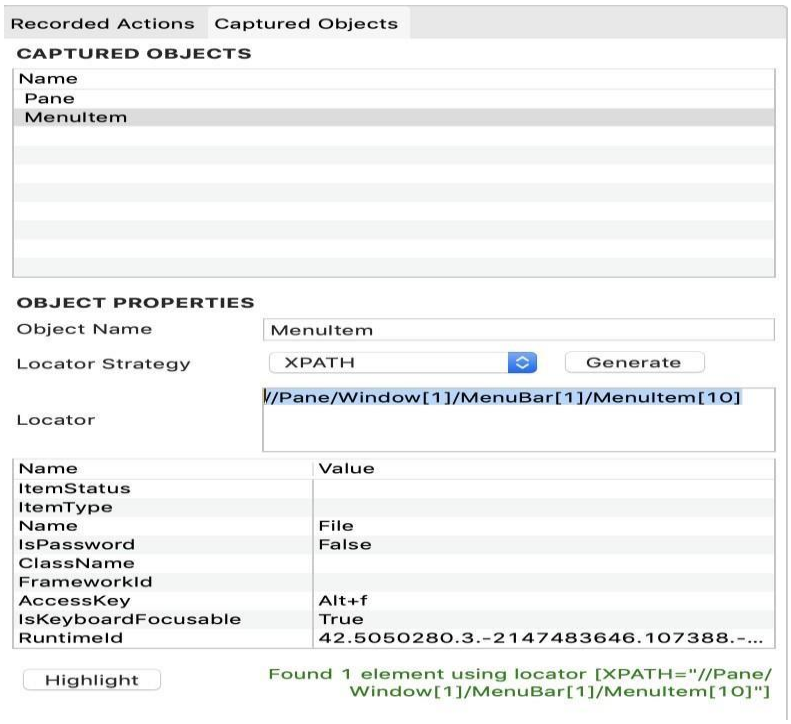
Configurations: Configuration: where you can view and edit the WinAppDriver URL and Desired Capabilities.
Application File: the absolute path to the Windows Executable File (*.exe) of the testing machine. For Windows users, click Browse... button to locate the application file.

The Start button is enabled after the Application File text box is filled.

Recorded Actions: The Recorded Actions part shows all of the recorded built-in actions for interacting with the application.

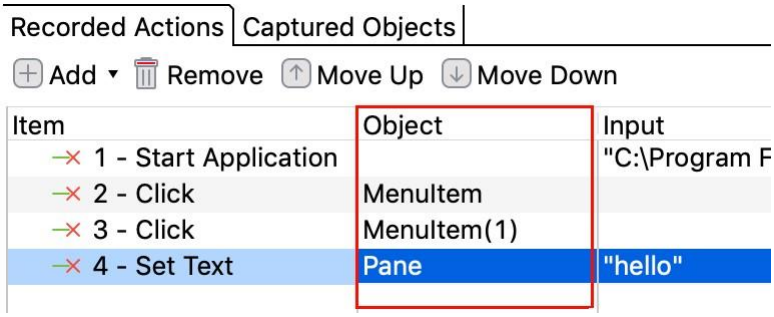


Captured Objects: The Captured Objects part stores all objects captured during the recording session. You can view and edit the object's name, locator and properties.

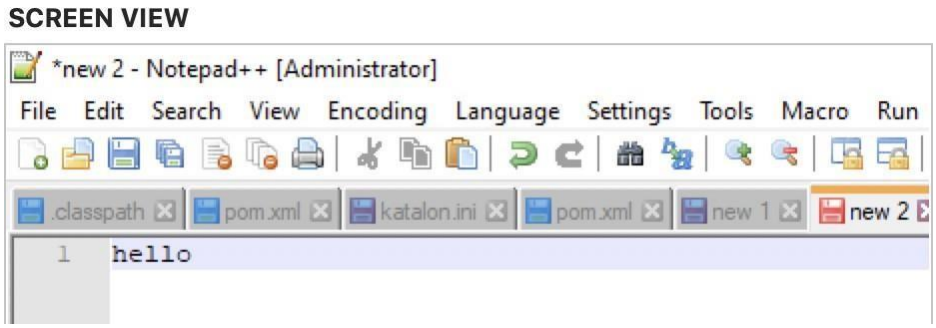


To customize the locator of a captured object, you can modify its locator in Object Properties and click Highlight to verify if the new locator correctly identifies the intended object.

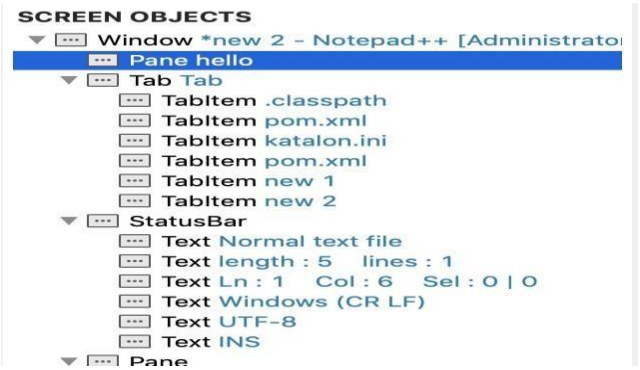
- In the Recorded Actions, you can also view the captured objects in the Object column.



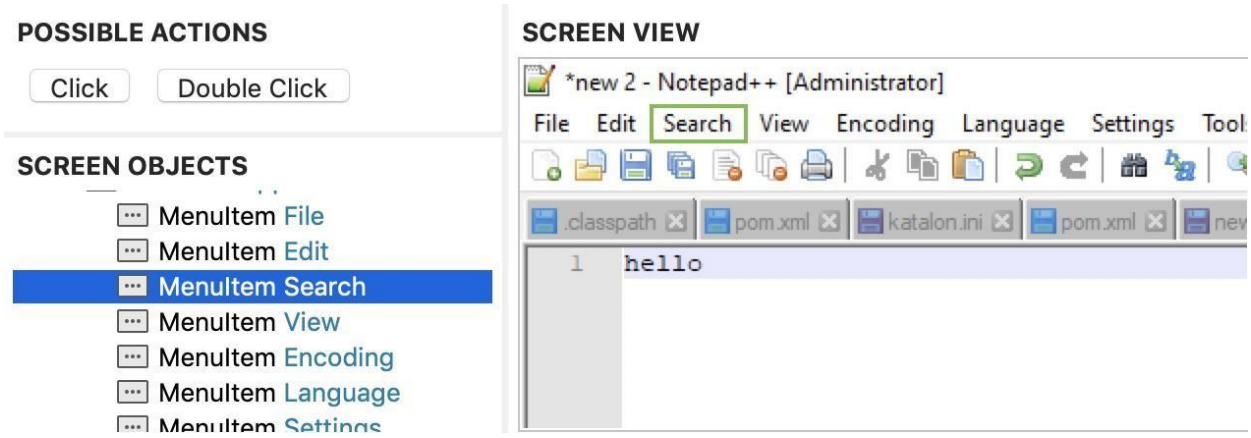
- When you press the OK button at the bottom of the dialog box to finish the recording session, you will be asked to store all captured objects in the Object Repository.
- Screen View
- The Screen View is a panel showing the application screenshots that are taken when you press either the Start button to record or the Refresh Screen button to recapture the application images. Those screenshots and the application on a real machine are of the same size.



Screen Objects: The Screen Objects section shows a tree of all objects that you can interact with. This tree will be automatically refreshed after you press the Refresh Screen button.



- The selected object is highlighted on the Screen View with a green rectangle surrounding it.

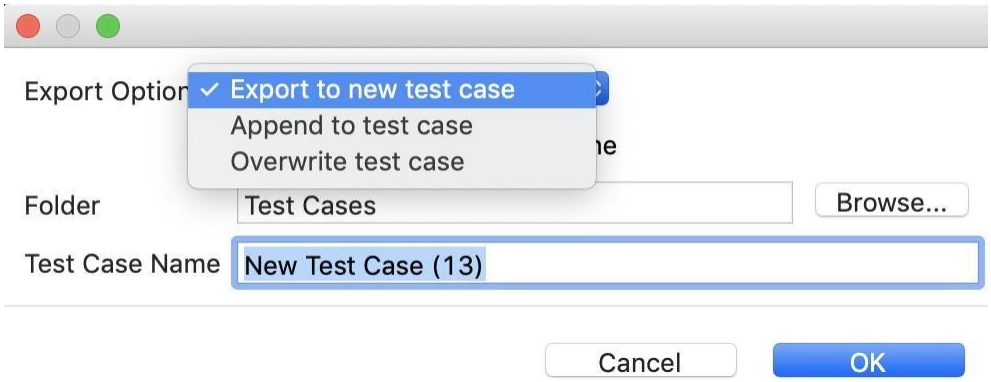


Possible Actions: The Possible Actions section shows a list of action buttons. Each button represents a Windows action.

- There are two types of Windows actions, including object action and application action.
- Object action: an action requiring you to select an object on the Screen Objects to perform the action.
- Application action: an action for interacting with the application.

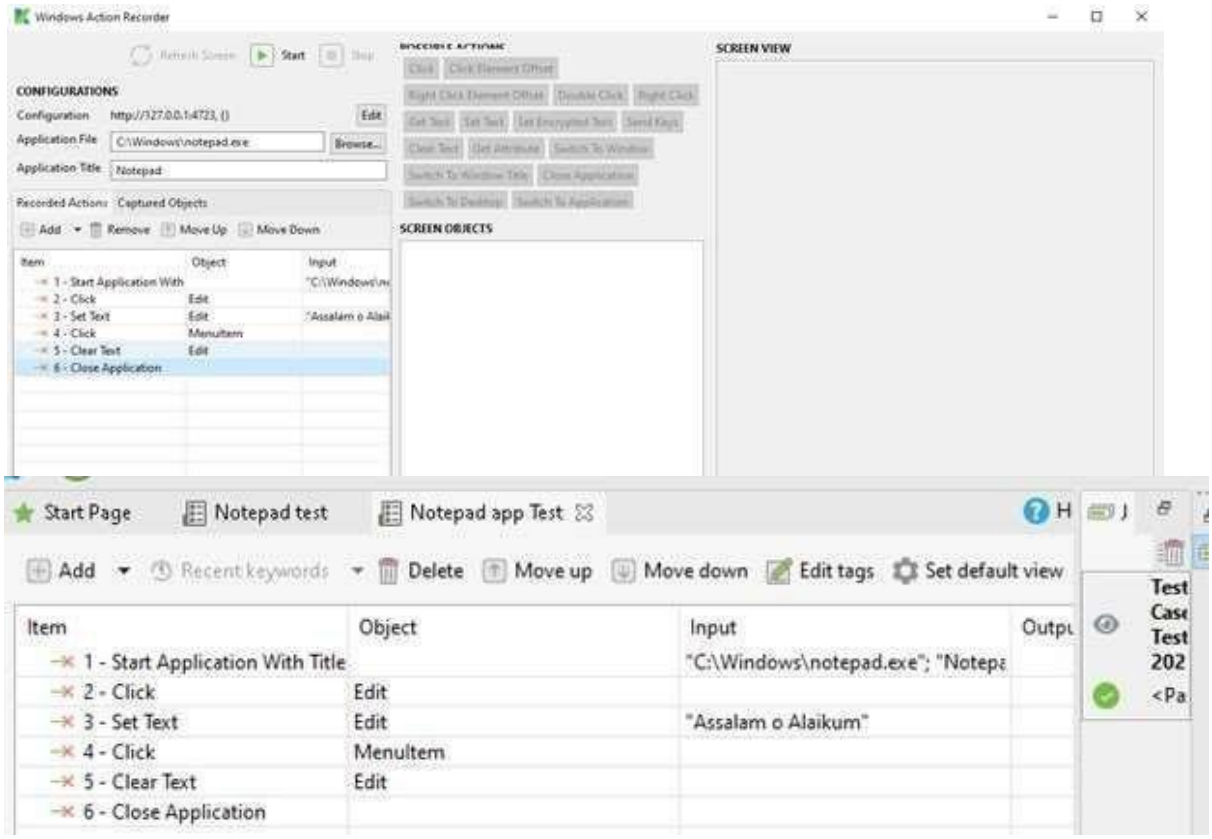
Export Option: When you finish the recording session, there are 3 options to export the recorded actions to a test case:

- Export to new test case (chosen by default).
- Append to test case
- Overwrite test case



EXERCISE: Test Notepad Application through katalon studio , *"write your name and also state the significance of katalon studio in software automation* "Attach all screenshot here

Screenshots



Test Summary Report:

Test Execution Log